SUBSET SELECTION IN HIERARCHICAL RECURSIVE PATTERN ASSEMBLIES AND RELIEF FEATURE INSTANCING FOR MODELING GEOMETRIC PATTERNS

A Thesis
Presented to
The Academic Faculty

by

Justin Jang

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
May 2010

SUBSET SELECTION IN HIERARCHICAL RECURSIVE PATTERN ASSEMBLIES AND RELIEF FEATURE INSTANCING FOR MODELING GEOMETRIC PATTERNS

Approved by:

Professor Jarek Rossignac, Advisor
College of Computing
Georgia Institute of Technology

Professor Karen Liu
College of Computing
Georgia Institute of Technology

Professor Greg Turk
College of Computing
Georgia Institute of Technology

Assoc. Professor Athanassios Economou
College of Architecture
Georgia Institute of Technology

Professor Charles Eastman
College of Architecture and
College of Computing
Georgia Institute of Technology

Date Approved: April 05, 2010

ACKNOWLEDGEMENTS

I wish to thank my advisor, Jarek Rossignac, for his guidance as a mentor and helping me develop as a researcher. His pattern as a diligent and faithful coworker has shown me how to be a valuable colleague, not just a collaborator.

I wish to thank my committee for their valuable feedback, suggestions, and advice in directing my research direction and leading me to ask the right questions about my topic.

I wish to thank Aaron Bobick for his practical and timely help with finishing my thesis.

I wish to thank my former advisors Bill Ribarsky and Chris Shaw. Both helped me learn how to do research and publish and both provided avenues for me to explore the field in a focused way.

I wish to thank the many staff members in the CoC, GVU center, and SIC; not just for their faithful and practical service, but for their courteous professionalism and their smiling faces.

I wish to thank my fellow labmates Jason, Brian, Topraj, Mark, and Tina with whom I spent many hours spread out over many semesters enjoying interesting conversations about almost anything. I would also like to thank the many other students and faculty with whom I have had the opportunity to attend group meetings, eat cake and ice cream, play ultimate frisbee, and win the school championship (twice).

I wish to thank my family whom I could always count on for unconditional support.

I wish to thank the saints for their prayers, fellowship, and portion of Christ.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

This thesis is concerned with modeling geometric patterns.

Specifically, a clear and practical definition for regular patterns is proposed.

Based on this definition, this thesis proposes the following modeling setting to describe the semantic transfer of a model between various forms of pattern regularity: (1) recognition or identification of patterns in digital models of 3D assemblies and scenes, (2) pattern regularization, (3) pattern modification and editing by varying the repetition parameters, and (4) establishing exceptions (designed irregularities) in regular patterns.

In line with this setting, this thesis describes a representation and approach for designing and editing hierarchical assemblies based on grouped, nested, and recursively nested patterns. Based on this representation, this thesis presents the *OCTOR* approach for specifying, recording, and producing exceptions in regular patterns.

To support editing of free-form shape patterns on surfaces, this thesis also presents the *imprint-mapping* approach which can be used to identify, extract, process, and apply relief features on surfaces. Pattern regularization, modification, and exceptions are addressed for the case of relief features on surfaces.

1 INTRODUCTION

The focus of this thesis is on modeling regular geometric patterns with explicitly designed irregularities in the context of parametric modeling. Let us first explore the notion of a regular pattern.

1.1 Regular patterns

Natural and man-made scenes exhibit regular and irregular patterns. For example, buildings often have many evenly spaced identical windows (Figure 1.1). Seats in an airplane are typically uniform in shape and arranged in straight rows (Figure 1.2). The possible arrangements are not limited to straight rows. For instance, a spiral staircase arranges stairs in a screw motion configuration (Figure 1.3). Sometimes, the patterns repeat themselves. For instance, the Mason Building at Georgia Tech has groups of four windows side-by-side (Figure 1.4). These groups of four windows are repeated in a row horizontally per floor and these rows of groups are arranged vertically across floors. Hence, one could surmise describing this as a pattern of a pattern of a pattern. Man-designed constructions do not always exhibit such rigid regularity. For example, many Antoni Gaudi and Frank Gehry structures combine freeform shapes with regular components and undulating compositions with straight or prescribed arrangements (Figure 1.5 and Figure 1.6). Finally, many works of art may possess forms and arrangements which range from being precisely regular to those which seem to completely defy formal structure (Figure 1.7). {Figure 1.8 and Figure 1.9 give additional examples of man-made scenes and objects which exhibit regularity in various ways.}

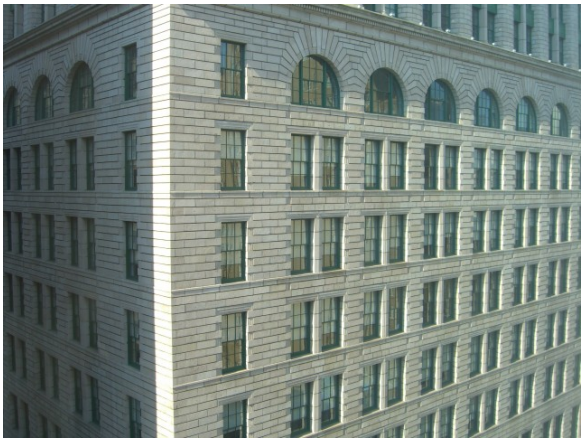

Figure 1.1: Buildings often have many evenly spaced identical windows.

Figure 1.2: Seats in an airplane exhibit uniformity in shape and are arranged in straight rows.

Figure 1.3: A spiral staircase arranges steps in a screw motion configuration.

Figure 1.4: Mason Building at Georgia Tech.

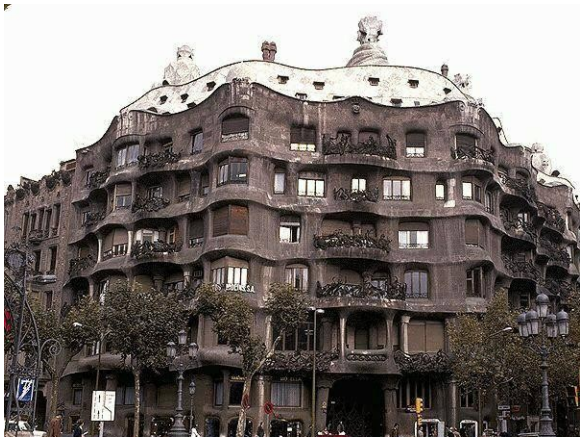

Figure 1.5: Antoni Gaudi's Casa Mila in Barcelona, Spain (image credit: www.greatbuildings.com).

Figure 1.6: Frank Gehry's Stata Center at M.I.T.

Figure 1.7: Art can exhibit various kinds of regularity or irregularity (image credit: Jarek Rossignac).

Figure 1.8: Man-made scenes exhibiting regularity.

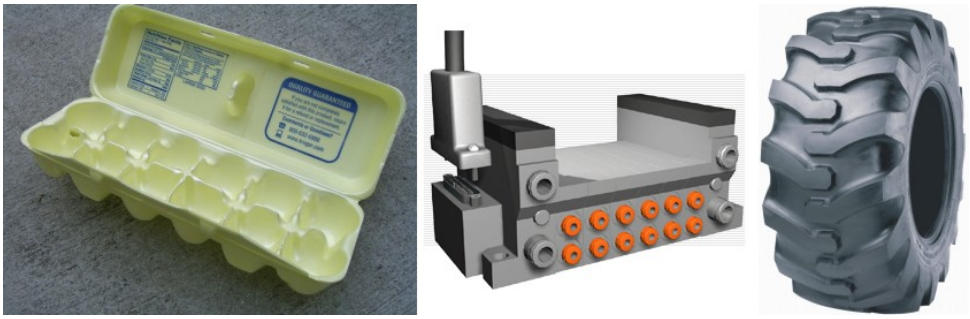

Figure 1.9: Man-made objects exhibiting regularity.

Nature also exhibits a wide variety of regularity. Branches on trees, petals on flowers, and leaf arrangements of ferns all exhibit some form of regularity and also some degree of irregularity (Figure 1.10). Even if perfectly identical forms and precise alignment do

not actually exist, we can plainly recognize that there is structure. Even in the animal kingdom, many examples of regularity exist such as the patterns on a snake and the segments on a caterpillar (Figure 1.11). In fact, many animal forms tend towards symmetry. One could even argue that the existence of some degree of regularity and order is part of the beauty of nature [Weyl 1982; Prusinkiewicz and Lindenmayer 1990].

Figure 1.10: Plants exhibiting regularity.

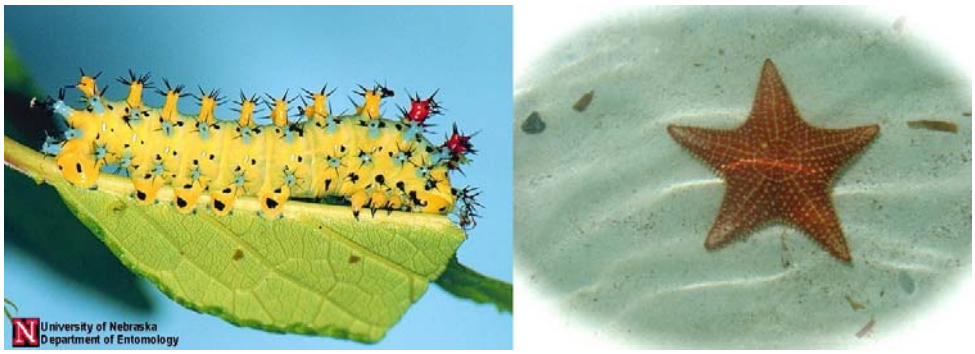

Figure 1.11: Animals exhibiting regularity – a Cecropia moth caterpillar (image credit: Jim Kalisch, Department of Entomology, University of Nebraska-Lincoln) and a starfish (image credit: Jarek Rossignac).

Further exploration of these notions begs for answers to the following questions.

- What is a pattern?
- When is it regular?

In this thesis we propose clear and practical definitions for patterns and pattern regularity. In particular, we characterize two kinds of regularity – regularity in shape (which we call ***feature regularity***) and regularity in arrangement (which we call ***frame regularity***). Hence, we say a ***regular pattern*** is a pattern that is both feature regular and frame regular (Figure 1.12). Furthermore, a pattern can be feature regular without being frame regular (Figure 1.13) or frame regular without being feature regular (Figure 1.14). Finally, a pattern can be neither feature regular nor frame regular (Figure 1.15).

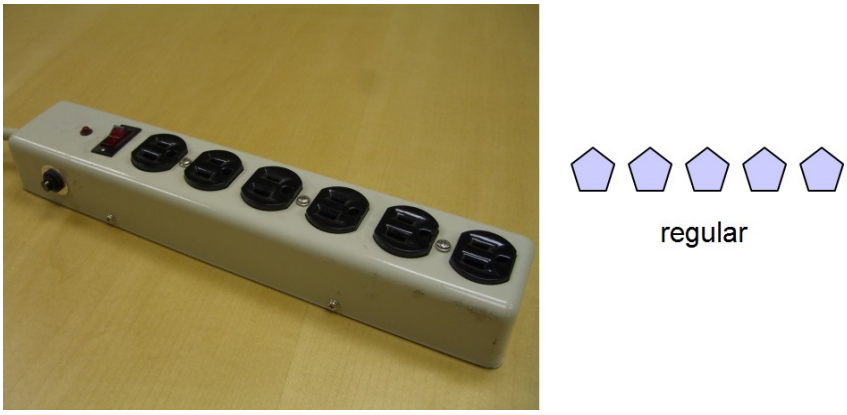

Figure 1.12: A regular pattern is regular in both arrangement (frames) and content (features).

Figure 1.13: A pattern can be feature regular without being frame regular.

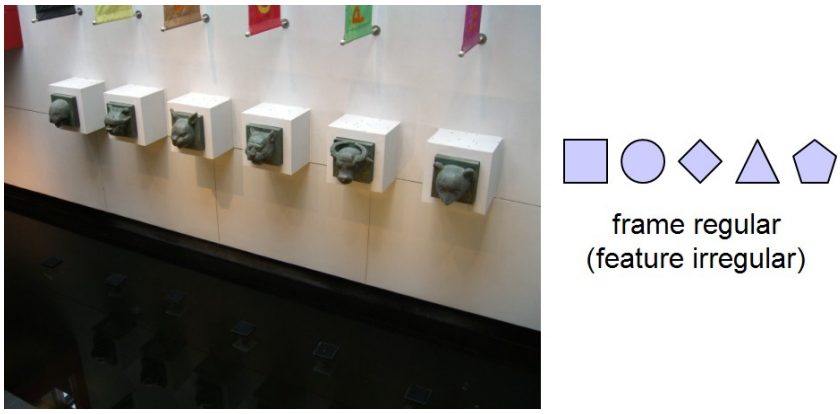

Figure 1.14: A pattern can be frame regular without being feature regular.

Figure 1.15: An irregular pattern is irregular in both arrangement (frames) and content (features).

In addition to these general notions of pattern regularity, we also propose that the components of a pattern have a particular ordering or sequence. This sequencing gives our definition practicality, allowing us to develop a coherent set of concepts that can be used to model a certain class of patterns. In particular, our definition allows us to describe and support the design and editing of hierarchical assemblies based on the ideas of grouped, nested, and recursively nested patterns.

1.2 Parametric pattern modeling

We now discuss pattern editing in the context of parametric modeling. To help us see some of the benefits of parametric modeling and, in particular, the benefits of the concept of a regular pattern in the context of parametric modeling, consider the following modeling scenario.

Suppose that a model was acquired with a 3D scanner. It has a regular pattern but it is not explicitly recognized. We may, for example, want to change the number of feature instances and change the spacing between them. A parametric model of a pattern would make such notions (e.g. number and spacing) explicit and thus facilitates editing of them

by simply adjusting parameters. Such capabilities can aid the designer in exploring the design space and can facilitate the creation of much content [vanderMeiden 2008]. However, without explicitly recognizing the regular pattern, the designer would be required to manually cut, paste, and delete feature instances to achieve edits which would typically be trivial in the context of parametric modeling.

Furthermore, because the model was scanned, it is imperfect and the pattern, while being nearly regular, is mathematically irregular. In the example in Figure 1.16, we are given such an irregular pattern of five cylinders protruding from the top of a larger cylinder in a nearly circular arrangement. To support editing in such a scenario, we identify four main steps: RECOGNITION, REGULARIZATION, MODIFICATION, and EXCEPTIONS.

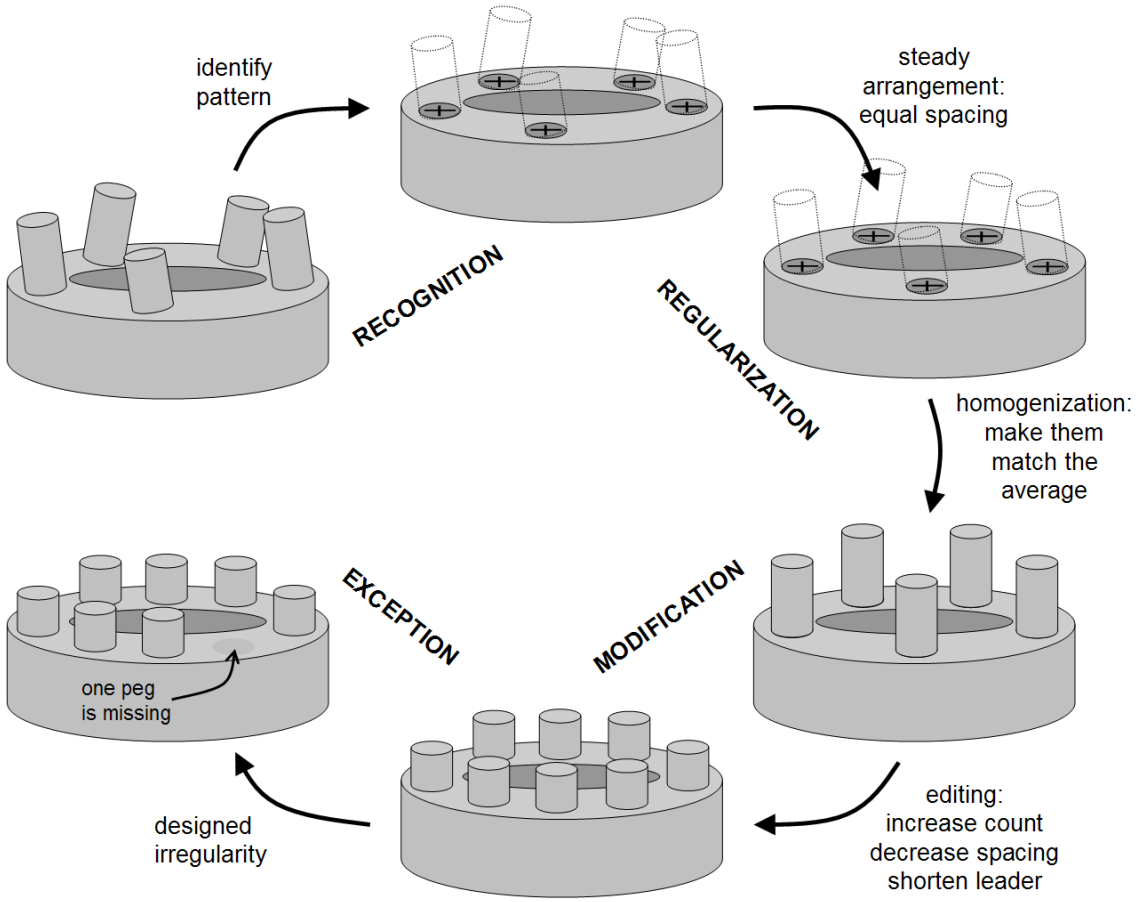

Figure 1.16: Pattern editing scenario.

The first step to editing a pattern here is to identify the pattern of cylinders and represent it as a pattern. We call this step RECOGNITION.

Once the pattern is identified, we would like to adjust the positions of the cylinders to make the nearly circular arrangement into a perfectly circular arrangement with equal spacing and orientation. We call such an operation FRAME REGULARIZATION. We would also like to make each cylinder in the pattern identical by changing it to match the average of all the cylinders in the pattern. This operation is commonly called beautification [Langbein 2003] but we refer to it as FEATURE REGULARIZATION to maintain consistency with our notions of pattern regularity. These regularization

operations, applied in any order, results in a pattern with a regular arrangement (frame regularity) and regular geometry (feature regularity).

Now we can edit the regular pattern by increasing or decreasing the number of cylinders and adjusting the spacing between them. We can even edit the shape of all the cylinders simultaneously by having changes made to one propagate to the rest. We may refer to such editing operations collectively as the MODIFICATION of a regular pattern.

To explicitly indicate a change to just one of the cylinders, we define an EXCEPTION by first making a selection of it and then modifying what we selected. An exception indicates an explicit deviation from regularity, hence, an "exception" to the "regular rule".

The result of such a pattern editing process is a model that has the potential to be substantially different in shape and yet embodies the basic semantics and design intent of the original.

In the given pattern editing scenario, we can see how the notion of a regular pattern can be a useful abstraction for editing models with repeated elements. A modeling operation which uses the concept of a regular pattern to generate content has the benefit of eliminating laborious repetitions from the design process. This is vital, especially where the repetitions are numerous, multiplicative (in the case of nested patterns), or exponential (in the case of recursively nested patterns). Yet sometimes the semantics of this concept are not stored in the model representation and this information on the design intent is lost. (This includes not only scanned models but also models from other sources.) Consequently, subsequent editing can be repetitive and laborious, even to the extent where modification is more expensive than redesign. This gives us motivation for

obtaining or extracting semantics which match the design intent. (Note that strictly speaking, if the original design intent was not recorded, the designer is guessing what the original design intent was or applying new intents.) This extra information along with the model in its existing realization allows the benefits of the pattern abstraction in the context of parametric modeling to be realized. Given a recognized pattern, feature and frame regularization can be applied to make the pattern a regular pattern. The model can then be edited as a regular pattern. For example, the designer can change the number of instances or the spacing between many instances by modifying a single parameter or transformation. Furthermore, the designer may wish to edit an explicit subset of the feature instances in a hierarchical pattern and maintain those modifications. Such changes to a subset of feature instances in a regular pattern signify an ***exception***, that is, some instances are explicitly designed to deviate from regularity. Hence, we would also like to support exceptions in hierarchical patterns within the framework of parametric modeling.

1.3 Contributions

In this thesis we make the following contributions.

A clear and practical definition for regular patterns is proposed. Based on this definition, we describe a representation and approach for designing and editing hierarchical assemblies based on grouped, nested, and recursively nested patterns.

Based on this representation for hierarchical pattern assemblies, we present an approach for specifying, recording, and producing exceptions in regular patterns.

To support editing of free-form shape patterns on surfaces, we address how to identify, extract, process, and apply (instantiate) relief features on surfaces.

Pattern regularization, modification, and exceptions are addressed for the case of relief features on surfaces. In doing so, we demonstrate that our regular pattern definition applies to different model types and representations, as long as the two basic elements are present: instantiation (of features) and arrangement (of frames).

Note that the contributions listed here are general. We provide a more specific and detailed list of contributions in the introductions for PART I and PART II of this thesis.

1.4 Organization

We organize this thesis into two parts. In PART I we give a definition for regular patterns, describe a representation and approach for designing and editing hierarchical assemblies based on grouped, nested, and recursively nested patterns, and describe how to specify, record, and produce designed irregularities (exceptions) in regular patterns including grouped, nested, and recursively nested patterns. In PART II we focus on a specific kind of feature, relief features on surfaces. In particular, we address the problem of relief feature transfer and describe an approach to copy, delete, cut, paste, move, and slide relief features on surfaces. Finally, in the context of relief feature patterns on surfaces we discuss how to make an identified pattern regular, edit it, and specify exceptions.

PART I and PART II complement each other in that PART I focuses on the arrangement (defining a set of frames) and semantics of regular parametric patterns while PART II focuses on the geometric content (to instantiate relief features given a set of frames) of regular parametric patterns. Specifically, we discuss exceptions in PART I and relief feature processing in PART II.

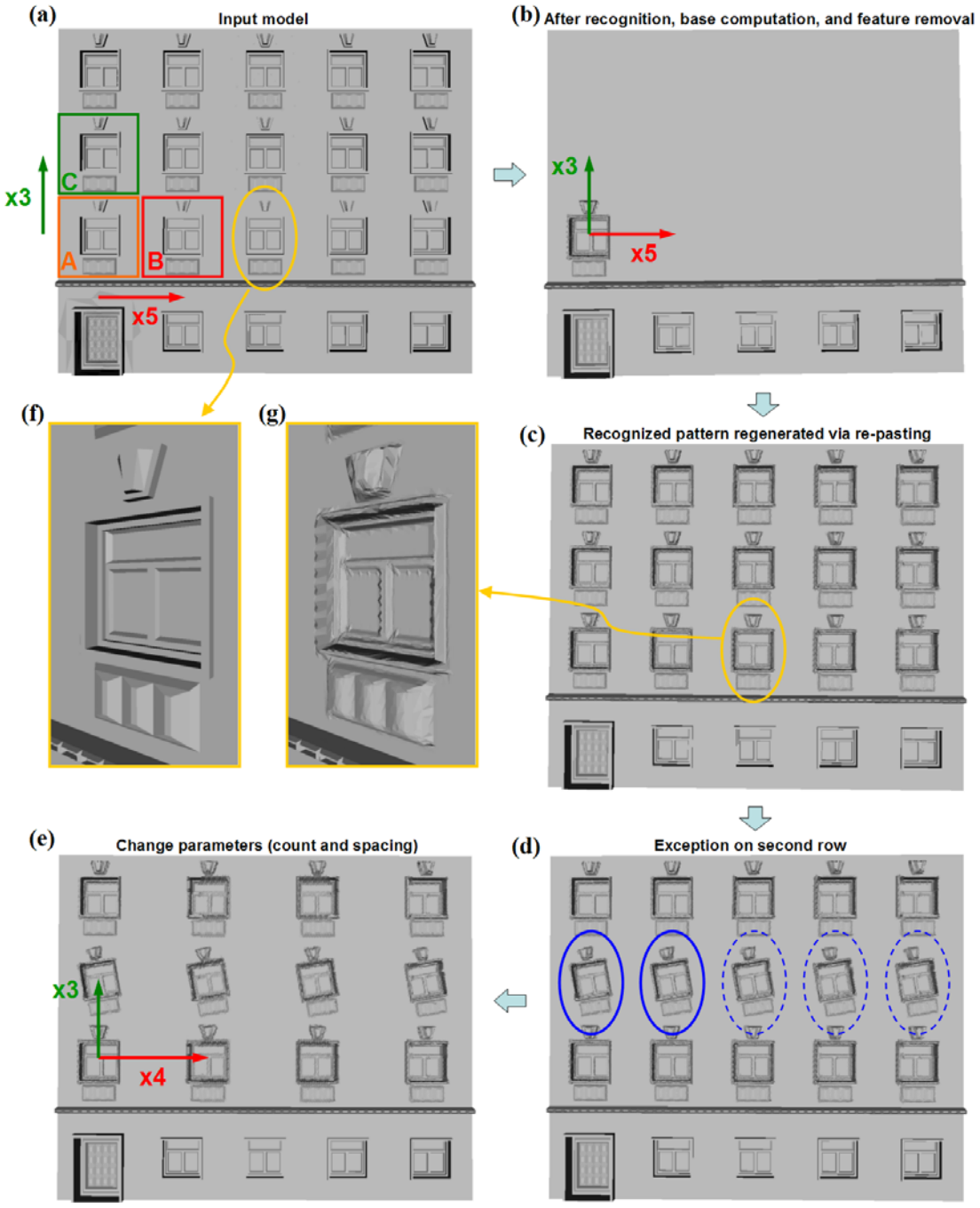

Figure 1.17: (a) The designer identifies three windows A, B, and C on the original input façade and indicates two counts (5 and 3). (b) The system recognizes the 5x3 pattern of relief features using the relative transforms between A&B and A&C respectively. A base surface is computed, the window feature relief extracted and stored as an imprint (a height field image), and the windows are erased. The first window is re-pasted from the imprint. (c) The entire 5x3 pattern is re-pasted. (d) To specify an exception, the designer picks two windows (solid ovals) and the system selects the whole second row. The designer rotates and shrinks one window and the rest of the selected windows are automatically updated by the system. (e) The designer changes the pattern parameters (count and spacing) and the system modifies the geometry accordingly, maintaining the exceptions. (f & g) Closeup oblique views compare an original and re-pasted window.

We illustrate how the two parts can be used together to support the re-design of models with patterns using the following modeling scenario. The building façade model in Figure 1.17 is an example output from a procedural facade generator [Wonka et al. 2003]. This façade model was processed to make it manifold in the upper section containing the 5x3 pattern of windows and the resulting mesh was subdivided using midpoint subdivision. The resulting input façade model is depicted in Figure 1.17a.

Now, the designer would like to edit what appears to be a 5x3 two-parameter pattern (nested pattern) of relief features. The designer specifies three camera frames (Section 3.1) A, B, and C by navigating the graphical view (Section 10.1). The system generates a nested pattern with the relief feature window identified by A as the pattern leader and with two pattern transforms (Section 3.9 and Section 3.10). One pattern transform is the relative transformation between A and B and the other pattern transform is the relative transformation between A and C (Section 4.1.3). This pattern of pattern of camera frames identifies the relief feature instances (the windows) in the pattern (Section 9.4). A base surface is computed (Section 9.5.2 and Section 10.3) and the windows are deleted by replacing them with the base (Section 9.5.5 and Section 10.5). The first window on the first row is re-pasted resulting in the model in Figure 1.17b.

The entire 5x3 nested pattern of windows can be regenerated by re-pasting the pattern leader window in all of the original positions as determined by the 5x3 nested pattern of camera frames (Figure 1.17c). The regenerated pattern of windows is similar to the original pattern of windows except that the windows have been captured into an imprint

(Sections 9.3, 9.5.1, and 10.5), erased, and re-pasted in place (Section 9.6). Hence, the resulting geometry is not identical due to the re-sampling (Figure 1.17f and Figure 1.17g).

Now the designer would like to specify an exception. To do this, the designer picks two windows on the second row (solid ovals in Figure 1.17d) and our selection system selects all of the windows on the second row (Sections 6.3 and 6.4). The designer then rotates and shrinks one window and the entire selection undergoes the same change (Figure 1.17d).

Finally, the designer can change the pattern parameters. For instance, the number of windows in a row is changed to four and the respective window spacing is increased. The existing exceptions are maintained (Figure 1.17e).

This example modeling scenario shows how the techniques and approaches presented in this thesis can be used to help a designer make and visualize design changes without knowing about the underlying structure and semantics which were originally used to generate a model.

PART I

2 PROBLEM STATEMENT AND MOTIVATION

In PART I we focus on pattern arrangements. That is, we discuss terminology, representation, and design of pattern assemblies without being limited to any specific shapes, types, or representations of geometry. In particular, we discuss how to define the pattern assembly semantics and parameters. That is, we discuss which and how many geometric components there are and the relationships between them. We apply these semantics and parameters to a specific class of pattern features in PART II, and further discuss issues related to the content of the pattern features in addition to arrangements.

To make the scope of PART I clearer, consider a circular pattern of pegs at the end of a pipe (Figure 2.1). We can model the complementary female pipe connection with the same circular pattern configuration but with a different feature, a thru hole instead of a peg. On the other hand, we can model studs of a diamond bracelet with the same pattern configuration. In fact, we can model the windows on a castle tower, the chairs around a breakfast table, the columns of a rotunda, and the spokes on a bicycle wheel with the same pattern. We can adjust the number of pegs, the angle spread between them, and the radius of the spread. Furthermore, we can adjust the relative twist or orientation of the pegs in their locations. From these examples, we can see the possibility of modeling many different models using not only the same concept, but also the same representation of the pattern. This is in line with one of the main benefits of parametric modeling, allowing versatile design by representing models using concepts and parameters, which can be readily adjusted, instead of the final realization [Shah and Mantyla 1995; ProE].

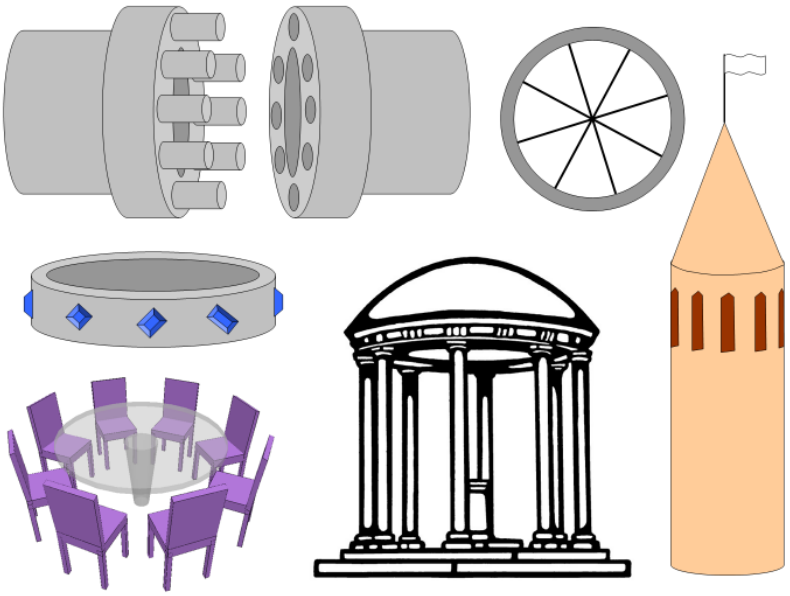

Figure 2.1: Models containing patterns which can be modeled with the same arrangement but different features.

These examples also hint at another benefit of procedural or summarized representations of patterns in the context of parametric modeling. A simple description can portray many elements. In fact, we show how a combination of several simple concepts (patterning/nesting/recursion, grouping) can describe and be used to generate multitudes of model elements. These concepts, in enforcing certain organizing constraints which reflect designer intent, can make the editing process more efficient, intuitive, and particularly adept at managing change. Again, such a parametric model can aid the designer in exploring the design space.

On the one hand, modeling repetition with patterns gives us an appropriate handle for modeling regularity. On the other hand, we need a way to support irregularity. Aside from stochastic and parametric approaches to irregularity, which operate on the set as a whole, we want to support exceptions in an explicit subset of pattern instances. In order to manipulate some subset, the designer needs a way to select a subset of instances, that is,

to make a selection. Hence, we also show how to support subset selection and further, to retain the specification of these exceptions through subsequent edits in models built with pattern hierarchies.

In summary, we answer the following questions in PART I:

-What are patterns?

-How can we represent patterns?

-How can we create and edit models with patterns?

-How can we efficiently select subsets in patterns?

-How can we record these selections so that they are persistent when parameters change?

We organize PART I as follows. Chapter 3 defines our terminology and describes the basic concepts underlying our approach. Chapter 4 gives prior art related to describing, representing, and editing patterns and arrangements of modeling components. In Chapter 5 we describe our representation for hierarchical patterns and give example patterns using our representation. Finally, in Chapter 6 we present our OCTOR approach to defining designed irregularities called exceptions by facilitating selection of subsets of objects in a hierarchical recursive pattern assembly.

3 TERMINOLOGY AND CONCEPTS

We now introduce our terminology and give our definition for regular patterns and more complex assemblies of multiple patterns.

3.1 Frame

Three linearly independent vectors (I, J, K) and a point (O) form a *frame* {I, J, K, O}. The idea of a frame is commonly used in computer graphics to represent a local coordinate system for a geometric component in a scene or assembly [Mortenson 2007; Gomes et al. 1998].

This way of defining a frame has a convenient geometric interpretation. The center O is the image of the global origin and hence captures the translational part of a transformation. The basis vectors I, J, and K indicate the orientation (rotation), scaling, and shear.

This geometric interpretation can be illustrated with a series of examples (Figure 3.1). Suppose the local coordinate system of a chair model is defined by frame {I, J, K, O}. When I, J, and K are mutually orthogonal and each have length one, the frame forms an orthogonal basis and the chair model retains its original scale and shape. Without invalidating this property, the chair model can be translated by changing O and rotated by rotating at least two of the basis vectors I, J, and K such that they remain orthonormal. If all three basis vectors I, J, and K are scaled to the same length, the chair model is uniformly scaled, i.e. changes size but retains proportions. If at least one of the basis vectors I, J, or K have a different length from the others, the chair model undergoes non-

uniform scaling, i.e. stretch and squash. If I, J, and K are not mutually orthogonal, the chair model will exhibit shear. Shear is indicated by the angles between the basis vectors. When the basis vectors I, J, and K are mutually orthogonal, the frame has no shear.

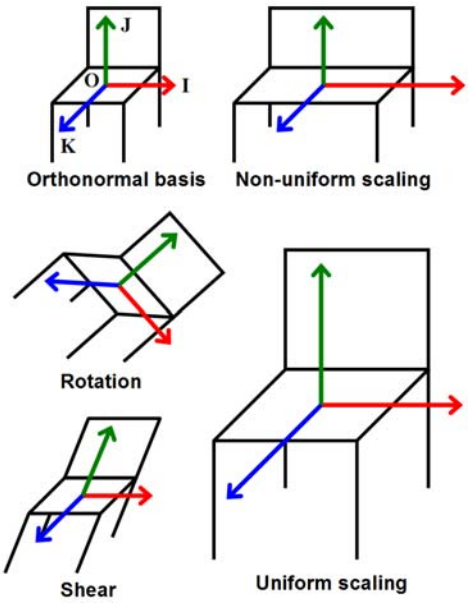

Figure 3.1: A frame {I, J, K, O} defines a local coordinate system for a chair model.

3.2 Affinity

An *affinity* is a transformation that preserves lines, parallelism, and proportions along lines. An affinity may be represented by the frame that is the image of the generic frame {**i**, **j**, **k**, **o**} = {(1,0,0), (0,1,0), (0,0,1), (0,0,0)} or by a homogeneous 4x4 matrix. The homogeneous form is useful for combining the effect of several transformations into a single matrix. The affinity T that maps one frame onto another can be defined by 4 pairs of matched points in general position T(A)=A', T(B)=B', T(C)=C', T(D)=D'. An affinity is always invertible. Identical to frames, two common special cases include similarity

affinity, which allows for uniform or non-uniform scaling, rotation, and translation but not shear, and Euclidean (rigid) affinity, which allows for translation and rotation but not scaling or shear.

3.3 Shape

A *shape* is a geometric model or part of a geometric model. A shape may be represented with respect to its own local frame. Then, to place and orient a shape is to place and orient its local frame.

Shapes are often defined implicitly in their own local frame for simplicity. For example, a unit cylinder could be defined parametrically as $C(r, h) = \{x,y,z \in R^3: ((x^2 + y^2 \leq r^2) \cap (z = 0 \text{ or } z = h)) \cup ((x^2 + y^2 = r^2) \cap (0 \leq z \leq h))\}$ (Figure 3.2). This cylinder uses the generic frame by construction.

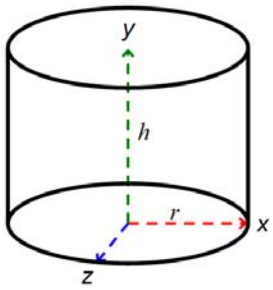

Figure 3.2: Parametric cylinder C(r, h).

In our terminology, a shape can be simple, complex, or compound. A simple shape can be a single primitive such as a sphere, cylinder, cone, box, regular polyhedron, or torus. Simple shapes also include manufacturing or assembly features such as slots, holes, or thru holes in feature-based CAD modelers.

More complex shapes include B-splines, NURBS, evolution surfaces, extrusions, polygonal meshes, and simplicial complexes in general including points, edges, and faces.

Compound shapes include any representation incorporating multiple geometric entities. Hence, representations such as BSP and CSG can be used to describe a model which is semantically considered a shape, e.g. in its final realization. In particular, compound shapes include patterns of shapes, a notion this thesis explores and formally encapsulates in defining the concept of a *feature*. Henceforth in this thesis, a shape (simple, complex, or compound) should be thought of conceptually as a single geometric entity.

3.4 Feature

A *feature* is a grouping mechanism used here to provide a recursive definition of assemblies. A feature may be a shape, an instance, an assembly, a pattern, or a recursive model as defined below. Actually, the idea of a feature is not the grouping mechanism itself but the wrapper which allows different kinds of components to be grouped. Hence, the concept of a feature supports the composition of compound shapes as well as simple shapes.

As an example, consider an assembly or a pattern that is composed of one or more features, each of which can be an assembly or a pattern. For instance, an assembly $A_1 = \{F_1, F_2\}$ of two features, where F_1 is a table shape S_1 and F_2 is a pattern P_1 of feature F_3. F_3 is a chair defined as an assembly A_2 consisting of six features F_4, F_5, …, F_9 which are a chair seat shape S_2, a chair back shape S_3, and four copies of leg shape S_4 (Figure 3.3). This illustrates how the concept of a feature generalizes the kinds of items that can be combined to form a model.

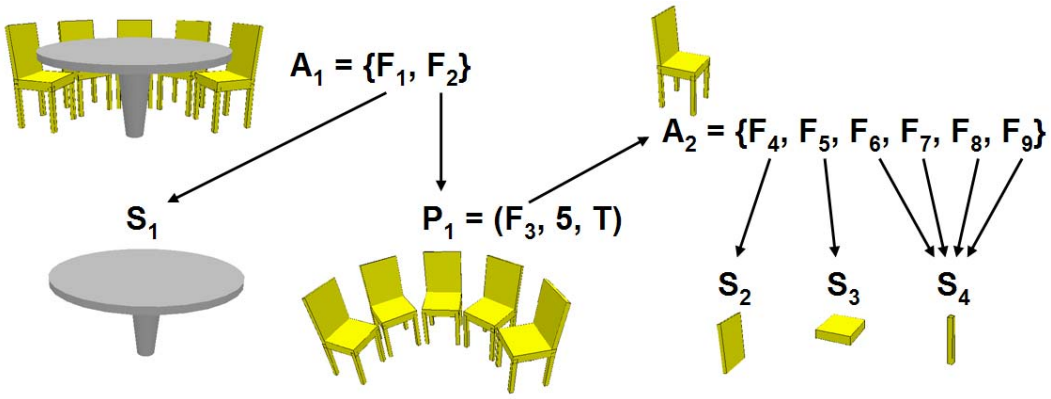

Figure 3.3: Using features as a grouping mechanism to create a compound assembly.

3.5 Instance

An *instance* is a feature F (typically a shape) transformed by an affinity. Accordingly, we define an instance as an association of a feature with a frame. In particular, an instance is the image T(F) of a feature F by an affinity T. To make things precise, we use the following notation. An instance will be written as (F, T). When cascading the notation, ((F, T), T') stands for T'(T(F)). The frame is used to place the shape in the global frame, that is, to *instantiate* a shape.

For example, to make the unit sphere, S_U ($PO^2 = 1$), appear as if it was sitting on the ground plane y=0, a frame with center O = (0, 1, 0) could be used. This corresponds to an affinity {**i**, **j**, **k**, O} which can be represented by the homogeneous 4x4 matrix T = {(1,0,0,0), (0,1,0,1), (0,0,1,0), (0,0,0,1)}. Hence the sphere would be instantiated as (S_U, T).

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The concept of instantiation also allows the set of primitives provided in a modeling package to be simplified and generic, requiring less or no parameters. Consider a cylinder defined parametrically as $C(r, h) = \{x,y,z \in R^3: ((x^2 + y^2 \leq r^2) \cap (z = 0 \text{ or } z = h)) \cup ((x^2 + y^2 = r^2) \cap (0 \leq z \leq h))\}$. This parametric form could be equivalently defined by defining a unit cylinder $C_U = C(1, 1)$ and instantiating C_U using a frame that scales the shape: $C_{new}(r, h) = (C_U, \{r*\mathbf{i}, r*\mathbf{j}, h*\mathbf{k}, \mathbf{o}\})$, where $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$, $\mathbf{k} = (0, 0, 1)$, and $\mathbf{o} = (0, 0, 0)$, or equivalently instantiated as $C_{new}(r, h) = (C_U, T)$, where $T = \{(r,0,0,0), (0,r,0,0), (0,0,h,0), (0,0,0,1)\}$.

$$T = \begin{bmatrix} r & 0 & 0 & 0 \\ 0 & r & 0 & 0 \\ 0 & 0 & h & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.6 Assembly

An *assembly* is a set $\{F_1, F_2, \ldots, F_n\}$ of features. Here we use curly braces "{}" to denote aggregation. When cascading the notation, $(\{F_1, F_2\},T)$ produces the same result as $\{T(F_1), T(F_2)\}$.

Figure 3.4 gives an example of an assembly which incorporates the notions of shape, patterns, sub-assemblies, and instantiation. A dining set assembly is composed of two features: a table instance (S_1, T_1) and a pattern instance (P_1, T_2). The pattern P_1 uses a chair assembly A_2 as its repeating feature. The chair assembly $A_2 = \{F_4, F_5, F_6, F_7, F_8, F_9\}$

is composed of six features: a chair seat instance (S_2, T_4), a chair back instance (S_3, T_5), and four leg instances (S_4, T_6), (S_4, T_7), (S_4, T_8), and (S_4, T_9).

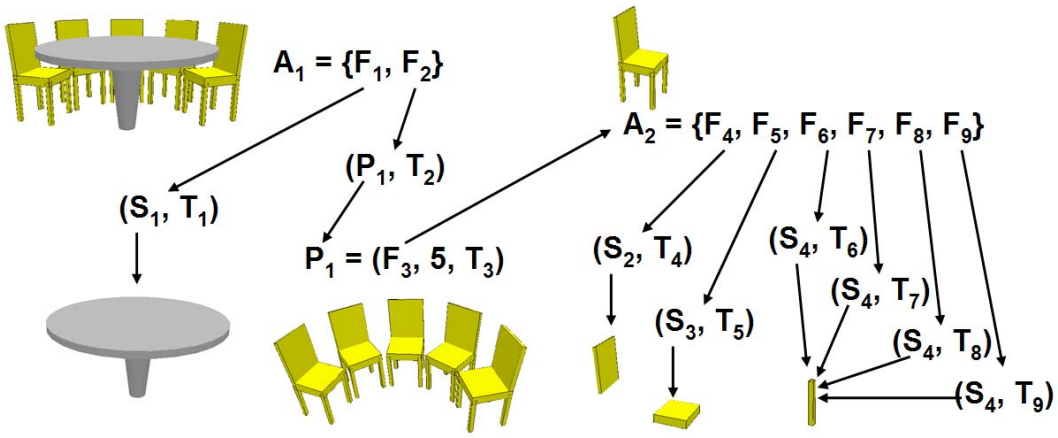

Figure 3.4: A compound assembly incorporating the notions of shape, patterns, sub-assemblies, and instantiation.

Next, we will discuss special assemblies. In particular, we discuss assemblies with properties suggesting some form of regularity.

3.7 Matching assembly

An assembly $\{F_1, F_2, \ldots, F_n\}$ of n features is *matching* when, given any pair F_i and F_j of features in the assembly, there exists an affine transformation $T_{i,j}$ such that $F_j = T_{i,j}(F_i)$. In other words, each shape in a matching assembly is the affine image of each other shape.

The features in a matching assembly do not need to be instances of the same original feature nor even instances of different shapes. They could be multiple representations of the same shape or be represented by a unique description.

Note that to be an instance is to be associated with an external frame (which could be the identity I), that is, to be transformed by an affinity which brings the local frame of the shape, assembly, or pattern to match the external frame. If the shape is not explicitly associated with an external frame which makes it an instance, the generic local frame applies. Hence, the definition of a matching assembly applies to assemblies of instantiated shapes, uninstantiated shapes, and a combination of instantiated and uninstantiated shapes.

We may generalize the notion of a matching set to include not just shapes as standalone entities but also features, each feature being an identified subset (cutout) of a model. Hence, a set of features may be identified on a single surface with a set of boundary curves and this set would be considered matching if each identified feature in this set is sufficiently similar to the others.

3.8 Pattern

A *pattern* is an ordered list, $F_1, F_2, \ldots, F_n$, of n features with a corresponding ordered list $T_{1,2}, T_{2,3}, \ldots, T_{n-1,n}$ of n-1 affinities which give the relative transformation between successive instances (F_i, T_i) and (F_{i+1}, T_{i+1}). Hence, a pattern is essentially an assembly with an ordering and a list of affinities.

A pattern of n features can be written as $(\{F_1, F_2, \ldots, F_n\}, n, \{T_{1,2}, T_{2,3}, \ldots, T_{n-1,n}\})$, which we call the general form for a pattern. While this could also be written without the count as $(\{F_1, F_2, \ldots, F_n\}, \{T_{1,2}, T_{2,3}, \ldots, T_{n-1,n}\})$, we prefer to write it with the count since the transformations $T_{i,i+1}$ do not represent instantiation transformations for corresponding features F_i but rather relative transformations between successive features F_i and F_{i+1}. Hence, there are n-1 transformations $T_{i,i+1}$ in a pattern of n instances.

Note that $\{F_1, F_2, \ldots, F_n\}$ does not need to be a matching assembly.

3.9 Regular Pattern

A *regular pattern* is an ordered list, $F_1, F_2, \ldots, F_n$, of matching features such that there exists an affinity T such that $F_{i+1} = T(F_i)$ for all $0 < i < n$. In other words, both the features and the frames are regular. A pattern will be written as (F_1, n, T), which we call the regular form. (See Figure 3.5.)

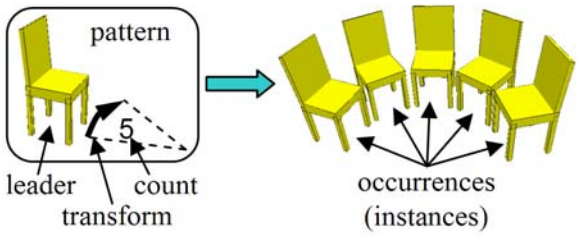

Figure 3.5: A simple regular pattern (leader, count, transform).

A pattern is *feature regular* when it uses a single feature F_1 as the unique prototype for all the features in the pattern. This unique prototype is called the *pattern leader*. When a pattern is feature regular then its pattern instances are matching.

A pattern is *frame regular* when its sequence of frames is steady. (Note that "frame regular" is called "steady" in [Rossignac and Vinacua 2009].) A sequence of frames is *steady* if they can be defined with a single unique transformation T such that $T_{i+1} = T_iT$; that is, each successive frame is defined as transformation T relative to its predecessor. In other words, $T_{i,i+1}$ are all the same for $0 < i < n$, where n is the *pattern count*. We call this unique transformation the *pattern-transform*. Since each frame is defined with respect to its predecessor, there needs to be an initial frame as a starting point. We call this initial

frame the *leader frame*. The leader frame is equivalent to the frame of the pattern itself. Hence, a pattern needs to be instantiated in order to define the frame of the first instance.

While a regular pattern is a pattern that is both feature regular and frame regular, a pattern can be feature regular without being frame regular or frame regular without being feature regular (Figure 3.6). When a pattern is feature regular (without being frame regular), the pattern can be written as (F, n, $\{T_{1,2}, T_{2,3}, \ldots, T_{n-1,n}\}$). When a pattern is frame regular (without being feature regular), the pattern can be written as ($\{F_1, F_2, \ldots, F_n\}$, n, T).

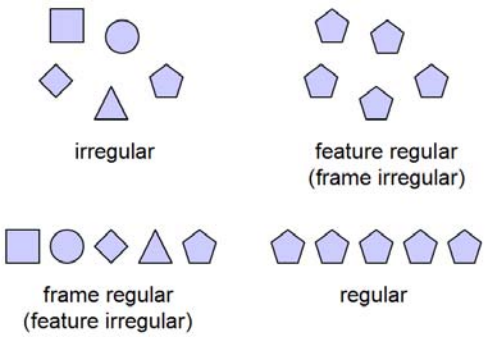

Figure 3.6: Classification of pattern regularity.

The process of making a pattern into a regular pattern is called *regularization*. The process of making a pattern into a feature regular pattern is commonly called *beautification* [Langbein 2003]; however, we prefer to use the term *feature regularization*. We use the term *frame regularization* to denote the process of making a pattern into a frame regular pattern. Hence, we can use feature regularization to make a pattern matching and we can use frame regularization to make a pattern steady.

3.10 Pattern of Patterns

We have defined a pattern as an ordered list of features, each of which has a frame defined relative its predecessor in the list. One implication of defining patterns as being composed of features is that patterns can be used to compose patterns. This occurs when the features in a pattern are themselves patterns. A pattern composed of a list of patterns is called a *pattern of patterns*. The patterns in the list are called the *child patterns* and the pattern of which these child patterns are members is called the *parent pattern*. In a pattern of patterns, a child pattern is used as the feature associated with each instance of the parent pattern. The frames of the child patterns are the frames of the parent pattern's instances. In other words, the parent pattern defines a list of leader frames for the child patterns. The parent pattern is a list of frames with child patterns being the features associated with the frames.

A *regular pattern of pattern* is a regular pattern with a regular pattern as the pattern leader. This results in a pattern which is essentially composed of one unique shape, the leader of the child pattern, and two unique pattern-transforms, belonging to the parent and the child patterns. This is modeled as a two-parameter pattern in some existing systems [IronCAD, Pauly et al. 2008], which also support the idea of n-parameter patterns.

Another way to say that a pattern is contained in the instances of another pattern is to say that a pattern is *nested* in another pattern. The child pattern C is nested in the parent pattern P. When the pattern leader of P **is** C, we say that C is *directly nested* in P. A *directly nested pattern* (DNP) is the feature that **fully defines** the pattern leader of the parent pattern. No additional components are included in the pattern leader. When the

pattern leader of a pattern (the parent P) **contains** a pattern (the child C) but other components are included, we say that C is *indirectly nested* in P. An *indirectly nested pattern* (INP) is contained in or is a component of the shape of the pattern leader of the parent pattern. Other components are included in the pattern leader. This inclusion is achieved through grouping (as an assembly). For instance, if C is the pattern leader of B then C is directly nested in B. If B is the pattern leader of A then B is directly nested in A and can be considered a DNP (Figure 3.7 top). Or, if B and D are grouped together as an assembly E which is the pattern leader of A, then both B and D are indirectly nested in A and we can say that B is an INP (Figure 3.7 bottom).

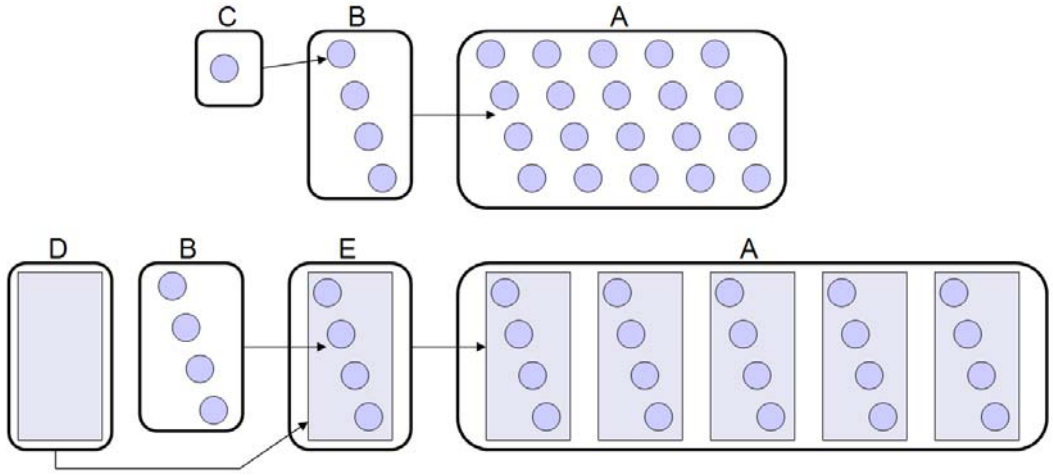

Figure 3.7: Directly and indirectly nested patterns. Top: C is directly nested in B. B is a DNP since it is directly nested in A. Bottom: D and B are grouped as E and E is directly nested in A. D and B are indirectly nested in A. B is an INP.

We make the distinction between DNP and INP to point out their differences in expressiveness. In particular, a DNP has only one unique pattern leader regardless of the nesting depth, while an INP has different pattern leaders at different levels of nesting and can include compound features composed of different shapes or features. Hence, between

the two there are differences in how to make a selection of a subset of features. An n-parameter pattern scheme as in [Pauly et al. 2008] and [IronCAD], which specifies the dimensionality of the nesting plus a transformation per dimension, supports DNP but not INP. In Chapter 6 we propose a selection scheme which supports both DNP and INP.

3.11 Recursive Pattern

Assume that we want to model a fern, e.g. Bransley's fern (Figure 3.8). Each branch consists of a main branch and a pattern of branches. Each sub-branch is recursively defined by the prototype definition of a branch. Hence, we can use a recursive pattern to model the fern.

Figure 3.8: Bransley's fern.

When a pattern P is composed of a list of features at least one of which contains a reference to pattern P, we say that P is a *recursive pattern*. In other words, if pattern P is nested under itself, then P is a recursive pattern. If pattern P is indirectly nested under itself, then P is a *recursive INP*. Likewise, if the pattern leader shape of P is a reference to the pattern P itself, P is *recursive DNP*.

In its basic definition, a recursive pattern is an infinite structure. Thus, recursion limits are placed in practice. The way these limits are enforced in practice depends upon the representation and implementation.

For a recursive model in general, we give a parameterized recursive definition of an assembly as R(d) = {… R(d–k)…}, where d is the recursion limit and k determines how d decrements toward the recursion limit. Parameter k will be 1 conventionally and 0 for an infinite recursive definition. Because of the recursive definition of a feature as a shape, instance, assembly, or pattern, the recursive parameter d also needs to be passed through a feature, shape, instance, assembly, or pattern. To terminate the recursive definition, we set R(0) = {} by convention.

For example,

$$F(n) = \{S, (P(n-1), T_1)\}$$
$$P(n) = (F(n), 2, T_2)$$

which can be written as one expression

$$F(n) = \{S, ((F(n-1), 2, T_2), T_1)\}$$

can be used to describe the “mouse ears” fractal model in Figure 3.9. S is a cylindrical disc.

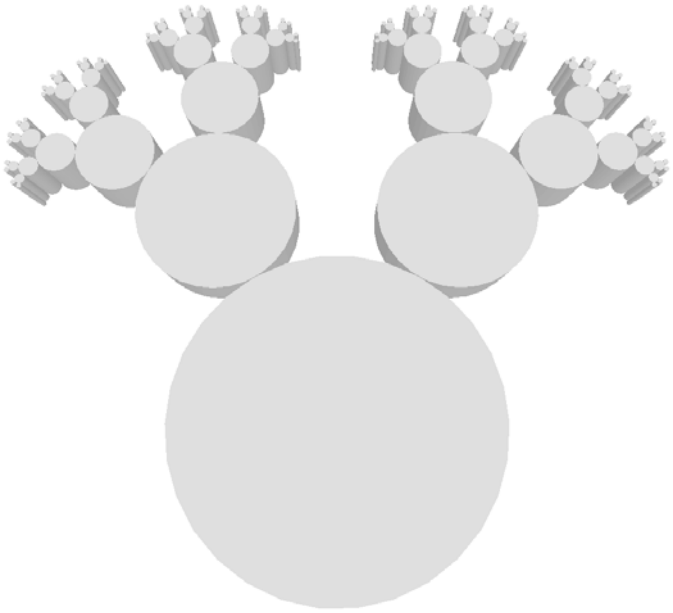

Figure 3.9: A recursive "mouse ears" model described by the expression F(n) = {S, ((F(n-1), 2, T_2), T_1)} invoked with F(6).

Any finite recursive model can be expressed as a non-recursive pattern assembly (including grouping) by expanding the recursion.

For example,

$$F(3) = \{S, ((F(2), 2, T_2), T_1)\}$$
$$F(2) = \{S, ((F(1), 2, T_2), T_1)\}$$
$$F(1) = \{S, ((F(0), 2, T_2), T_1)\}$$
$$F(0) = \{\}$$

Hence,

$$F(3) = \{S, ((\{S, ((S, 2, T_2), T_1)\}, 2, T_2), T_1)\}$$

Since $\{S, ((\{\}, 2, T_2), T_1)\} = \{S, (\{\}, T_1)\} = \{S, \{\}\} = \{S\} = S$.

Now let us point out one special case of recursive patterns. Let us define a *recursive loop path* as the set of expressions which need to be evaluated to resolve a recursive link back to itself. If this path is unique for each recursive link defined, then the recursive loop path can be represented as a single pattern. For example, F(n) = {S, ((F(n-1), 12, T_2), T_1)}

has 12 references back to F(), one for each instance in the pattern (F(n-1), 12, T_2) and thus cannot be recomposed as a single simple pattern. On the other hand, F(n) = {S, ((F(n-1), 1, T_2), T_1)} can be equivalently written as F(n) = {S, (F(n-1), T_1)} since the pattern of one instance (F(n-1), 1, T_2) is simply its pattern leader F(n-1). Then, F(n) = {S, (F(n-1), T_1)} = {S, ({S, (F(n-2), T_1)}, T_1)} = … = (S, n, T_1). (See Figure 3.10.)

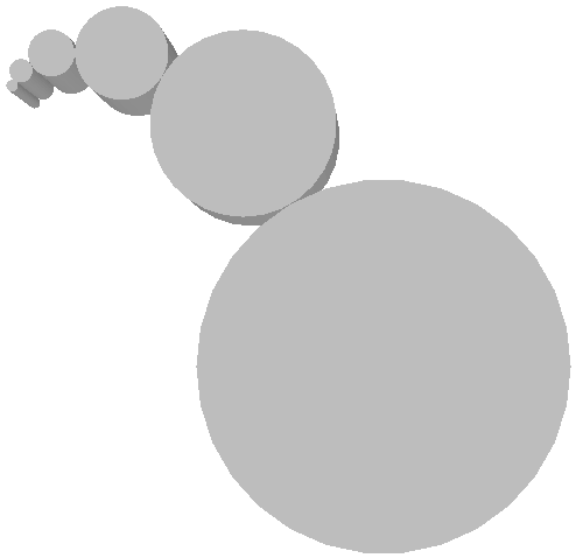

Figure 3.10: This single branching recursive model can be redefined as a pattern F(n) = {S, ((F(n-1), 1, T_2), T_1)} = (S, n, T_1). F(6) is shown.

3.12 Naming persistence

Once a model is defined, certain components and features may be identified for further editing. Attributes or constraints may be assigned to or new components may be added relative to them (the identified components). The challenge arises when the underlying model is modified by, for example, changing the position, dimensions, or some other attribute of the component. To maintain the assigned attributes, constraints, or

child components, the need arises to continue to identify or persistently *name* the same components.

For example, a pattern of five pegs is defined and a ring is defined to hang on the fifth peg. If the user increases the number of pegs, the description “fifth” is still valid. However, if the user decreases the number of pegs to four, the description “fifth” is naming an instance which is not defined by the pattern anymore. Now if the user decides to increase the number of pegs back to a count greater than five, the naming has persistence if it is retained and is once again valid.

3.13 Summary

We have given terminology and proposed a definition for regular patterns and more complex assemblies of multiple patterns. Given these definitions, we will describe a practical representation and approach for designing and editing assemblies composed of patterns with varying degrees of regularity and complexity.

4 PRIOR ART

Based on our definitions, the intrinsic content of an assembly is a set of frames coupled with a set of geometric components. Hence, we review three main topics in the prior art: affine transformations, scene/assembly representations, and naming persistence.

The set of frames is described as a *base frame* plus a set of *derived frames* obtained by applying a set of affinities to the base frame and any of its derived frames therein. Hence we discuss computing and applying relative transformations between frames.

We also need a representation to facilitate design, editing, and storage of assemblies. Hence we discuss various approaches to representing arrangements of geometric components in space.

Finally, in designing and editing models, the designer needs a way to identify or *name* logical components. However, changes during the design and editing process may affect the persistence of the names and the existence of the components. Hence, interpretation of the naming of the components requires processing when the pattern model is changed. We summarize the key prior art on the naming problem as it relates to our work.

4.1 Math background (transformations)

Here we give a basic background on transformations in three-dimensional space. In particular, we consider four problems concerning computing transformations: (1) how to aggregate a series of transformations, (2) how to linearly combine transformations, (3) how to compute a single relative transformation between two frames, and (4) how to compute a steady (defined in Section 3.9) series of transformations between two frames.

We limit our discussion to affine transformations (also known as affine maps or affinities), which can be composed of rotation, scaling, shear, and translation transformations. Furthermore, we explain the approaches in terms of matrices in homogeneous coordinates. As said in Section 3.2, an affinity can be represented by a 3x3 matrix for the linear transformations (rotation, scaling, and shear) plus a 3x1 vector for the translation component of the transformation. Using a 4x4 matrix in homogeneous coordinates allows rotation, scaling, shear, and translation to be combined into a single matrix. Furthermore, multiple affinities can be combined using matrix multiplication. Hence, when we for example mention an affinity A and a frame F in our discussion, we are referring to the 4x4 homogeneous matrices A or F depending on context.

4.1.1 Aggregating transformations

The first problem we consider is how to compute an affinity A that is equivalent to applying a series of n affinities $A_1 \ldots A_n$ in succession.

The effect of a series of n affinities $A_1 \ldots A_n$ can be aggregated into a single affinity by multiplying their respective matrices together:

$$A = \prod_{i=1}^{n} A_i \ .$$

We can apply this concept to frame regular patterns. For a frame regular pattern of n components using pattern transform T, the relative transform of the k-th component to the first component can be computed as:

$$T_k = T^{k-1} = \prod_{i=1}^{k-1} T \ .$$

4.1.2 Linear combinations of transformation

The second problem is how to support linear combination of transformations. In particular, supporting the linear combination of transformations provides the ability to created weighted combination of transformations, interpolate between transformations, and to construct or use arbitrary transformations in a structure similar to a basis of a vector space [Alexa 2002].

Alexa [2002] defines a scalar multiple operation and a commutative matrix combination operation which they call commutative addition. Together, these operations facilitate the linear combination of transformations (LCOT). A brief summary of the LCOT approach is provided here.

Scalar multiples. Intuitively, a scalar multiple $\alpha \otimes T$ should have the property that when it is applied $1/\alpha$ times the result is T. For instance, the scalar multiple of T with $\alpha = \frac{1}{2}$ applied twice should be equal to T, i.e. $(\frac{1}{2} \otimes T) \circ (\frac{1}{2} \otimes T) = T$. Note that multiplication $A \circ B$ is equivalent to commutative addition $A \oplus B$ (defined later) if A=B (which is the case here). The same should follow for one-third applied three times, one-fourth applied four times, and so forth. Alexa shows that for the cases of translation, rotation, and scaling the operation $\alpha \otimes T$ corresponds to T^{α}, which is valid for arbitrary real powers of a matrix when the matrix has no negative real eigenvalues. Scalar multiples may also be expressed using exponential and logarithm as $r \otimes A = e^{r \log A}$.

Commutative addition. Given two square real matrices A and B of the same dimension, AB and BA are different in general. However, by breaking A and B into smaller parts and multiplying their parts alternately, the difference in the resulting combinations $(A^{1/n}B^{1/n})^n$ and $(B^{1/n}A^{1/n})^n$ is less than the difference between AB and BA.

Taking this to the limit as n goes to infinity, Alexa defines the commutative addition $A \oplus B = \lim_{n\to\infty}(A^{1/n}B^{1/n})^n$. Commutative addition can be expressed using exponential and logarithm as $A \oplus B = e^{\log A + \log B}$.

Linear combination. A linear combination of an arbitrary number of transformations T_i with weights w_i is computed as $\bigoplus_i w_i \otimes T_i = e^{\sum_i w_i \log T_i}$. Note that if $A \oplus B$ was not commutative, $(1-t) \otimes A \oplus t \otimes B$ would not produce a steady (i.e. frame regular) pattern as the solution proposed in [Rossignac and Vinacua 2009] does.

4.1.3 Relative transformations

The third problem we consider is how to compute the relative affine transformation, the affinity A, between two given frames, T_1 and T_2.

Let the two frames T_1 and T_2 and the affinity A be represented by 4x4 homogeneous matrices. The two frames T_1 and T_2 are equivalent to affinities that map the global frame into the local frames T_1 and T_2. Hence, we can compute the affinity of T_2 relative to T_1 as $A = T_2 \cdot T_1^{-1}$, since affinities are invertible.

4.1.4 Extracting affinity roots

The fourth problem we consider is how to compute the n-th root of a transformation. In other words, given an affinity A, compute M such that $M^i = A$. Extracting the roots of an affinity has application in modeling (e.g. creating patterns) and in animation (e.g. computing a steady affine morph, a time-parameterized family of shapes). For example, given two poses represented by frames T_0 and T_n, where $T_n = A \cdot T_0$, we can compute intermediate frames $T_1 \ldots T_{n-1}$ and in particular, pattern transform $T_{0,1} = T_{1,2} = \ldots = T_{n-1,n}$

= M, where M^i = A. This affords the designer the flexibility to specify the pose of the final pattern instance, not just the option of specifying the pose of the first and second instance.

Rigid body motion. Rossignac and Kim [2001] describe how to compute pose interpolating 3D motions for rigid bodies (Euclidean motion) using screw-motions. A screw motion has the property that it is fully defined by the initial and final frames. It combines a minimum-angle linear rotation around a fixed axis of direction S with a minimum-distance linear translation along S. Furthermore, since this rotation around and translation along the same axis commute and thus can be linearly combined, we can compute a steady motion as the screw with 1/n of the rotation angle with 1/n of the translation distance.

Rigid body motion with scaling. Alexa [2002] shows that for the cases of translation, rotation, or scaling, the scalar multiple operation $\alpha \otimes T$ corresponds to T^{α}. Taking the n-th root means computing $\alpha \otimes T$ with $\alpha = \frac{1}{n}$. Hence, the scalar multiple operation can be used to extract affinity roots for rigid body motions with scaling.

Rigid body motion with scaling and shear. Rossignac and Vinacua [2009] present a solution to this problem called the Extraction of Affinity Roots (EAR) algorithm. The approach decomposes the set of affinities into complementary configurations and uses an explicit closed from solution for each configuration for which the logarithm of the linear part of the affinity exists. The resulting approach is able to compute a steady motion if it exists and use approximate solutions when no steady solution exists.

4.2 Scene/assembly representations

Here we discuss representations for arranging geometric objects in space. Such an arrangement of objects can be used to describe a model of a scene or an assembly. The basic problem is to provide a functional representation of all the frames corresponding to a set of geometric objects which comprise a model. The representation should facilitate the creation, design, and editing of the spatial arrangement of the objects in the model.

The prior art for modeling geometric arrangements in general is vast and diverse. Since we are interested in modeling arrangements with structure and regularity, in particular patterns, we limit our discussion to approaches which attempt to or could be used to represent patterns or "pattern-like" structures.

Furthermore, we are interested in an approach where the patterns are in some way compactly represented to summarize the content. The representation should provide access to the most relevant parameters related to patterns, e.g. how many instances there are in a pattern and how they are spaced, and facilitate straightforward modifications to the pattern when the parameters are adjusted. In other words, the representation explicitly stores relations and parameters describing the patterns modeled.

It is possible for the user interface to present the concept of a pattern to the user without the representation maintaining the concept in the data representation. For example, PowerPoint [PowerPoint] allows users to create a pattern of objects using the duplicate command. Specifically, the user duplicates the selected object, repositions the newly duplicated object, and then repetitively duplicates the same object. These commands must be performed in succession without executing other operations in between. However, the relationship between the original object and its duplications are

not stored. Hence, the user cannot change the count or re-space the objects without performing further delete, copy, and paste operations. Furthermore, in order to make a change in all of the duplications, the user must manually make the same change in each of the duplications or to make the change in one of them and manually duplicate the rest as before. Such repetitive manipulation may be suitable in certain applications where small-scale tasks are typical, but in general we would like to automate repetitive actions as higher-level operations.

There are many possible representations for geometric assemblies that go beyond a scattered collection of geometric components in a soup, for example, a single one-dimensional list of components with their frames. These representations express the geometric and semantic relationships between the individual components and subcomponents which comprise the assembly. A basic question that can be asked is whether a graph (typically a tree or a rooted directed acyclic graph (RDAG)) is needed. Hence, we can classify such approaches as either graph-based or non-graph-based. Graph-based approaches, which use an explicit graph to organize the components, fall under the general umbrella term *scene graphs* [OSG]. The different variations of such approaches are called *scene graph implementations*. Some non-graph-based approaches use procedural means to generate the content which captures the geometric and semantic relationships of the constituent components in a model. Hence, they are referred to as *procedural modeling* approaches [Ebert et al. 2003]. Procedural modeling approaches include shape grammars, L-systems, fractals, generative modeling, and modeling languages in general. Note that scene graphs can also incorporate procedural approaches, e.g. CSG and procedural geometric instancing (PGI) [Ebert et al. 2003], so they are not

exclusive to non-graph-based approaches. In addition, note that procedural approaches could also be used to generate graphs, but the graph itself would not be the persistent representation, but rather the commands which generated the graph. In other words, the primary representation is not the graph itself but the sequence of commands or "history" which generates the graph [Ganster and Klein 2007].

We now describe these modeling approaches in more detail, focusing on how they have been or could be used to model patterns. In particular, we have mentioned three main desired characteristics:

- The approach should represent **arrangements** of shapes; we will not discuss shape representations in general.
- The approach should represent **patterns**, not arbitrary arrangements in general.
- The approach should **explicitly represent** pattern relations and parameters.

4.2.1 Scene graphs

Scene graphs refer in general to data structures which arrange the logical relationships of the components in a model. Scene graphs can also be used to additionally represent the spatial relationships between components in a model. Thus they are often used to describe a scene with many objects or an assembly with many parts.

Because of the generality of scene graphs, they are used ubiquitously in applications such as vector-based graphics editing applications, CAD/CAM software, and computer games. Whether it is visible to the user or not, most graphical modeling and editing applications use some form of a scene graph to represent models, scenes, or layouts. This includes vector-based graphics editing applications such as AutoCAD [AutoCAD], Adobe Illustrator [Illustrator], CorelDRAW [CorelDRAW], and PowerPoint [PowerPoint]

and geometry modeling applications such as Maya [Maya], 3dsMax [3dsMax], Pro/ENGINEER [ProE], CATIA [CATIA], form-Z [formZ], IronCAD [IronCAD], SolidWorks [SolidWorks], NX [NX], Rhinoceros 3D [Rhino3D], and MicroStation [MicroStation]. In particular, vector-based graphics editing applications usually prefer to hide the graph from the user, while 3D modelers often make some form or some visualization of the graph readily available and may even rely on it as a fundamental means of interacting with the model and its structure. Regardless of the presentation, scene graphs in these applications represent logical and spatial semantics among the components of a model.

A common scene graph interpretation involves organizing nodes as a directed graph or tree. There are two main kinds of nodes, grouping (interior) nodes and leaf nodes. The main function of a grouping node is to logically group (usually disjoint) a set of child nodes, but there can be many other secondary functions such as appending transformations, injecting shading information, defining views, or providing spatial information for culling. Hence, part of the main function of a grouping node involves propagating such information to its child nodes. Allowing scene graphs to represent geometric content, the primary function of a leaf node is instantiation; that is, it points to a geometric primitive, shape, or entity. In summation, the interior nodes provide organization for the nodes which they aggregate and the leaf nodes indicate the basic content being organized.

One powerful characteristic of scene graphs is that child nodes inherit properties and parameters from parent nodes and operations (unary and n-ary) performed on parent nodes are propagated to child nodes. This idea has been ubiquitously applied to

transformations such that the pose (frame) of any single or compound object represented in the graph can be computed as the accumulation of transformations from the root to the node representing the object. Furthermore, the relative transformation between a node N and one of its ancestor nodes A is the accumulation of transformations from A to N. A practical benefit of a group of components inheriting transformations from their common parents is that a compound object may be moved as if it were a single object, which is consistent with the real-world interpretation of joined objects. Another common application of this feature is managing the articulation of joints, e.g. for the animation of limbs.

The idea of incorporating transformations into scene graphs can be extended to patterns. In fact, a number of 3D modeling applications support some form of simple pattern in a way that the pattern concept persists (is stored in) in the representation, e.g. IronCAD [IronCAD], AutoCAD [AutoCAD], Pro/ENGINEER [ProE], CATIA [CATIA], etc. These approaches are primarily limited to simple configurations such as one or two-parameter systems based on straight line or radial arrangements on flat surfaces and consisting of a simple pattern leader, but some have provision for patterns of arbitrary complexity using grouping. Grouping enables any number of constructs to be brought together and used as the pattern leader. In principle, this includes other patterns [van Emmerik et al. 1993], in which case n-parameter systems [Pauly et al. 2008] could be modeled using directly nested patterns (DNP), where patterns are pattern leaders, or more complexly nested systems could be modeled using indirectly nested patterns (INP), where patterns are contained in pattern leaders through grouping. (Section 3.10 defines DNP

and INP.) In addition, more complex arrangements can be specified, for example, by using reference curves to which successive pattern instances are constrained [ProE].

As with any paradigm, the way to support a pattern paradigm in scene graphs is to design a node with a specific interpretation. For example, van Emmerik et al. [1993] essentially interpret a pattern node such that the child node is treated as the pattern leader. The pattern node itself includes the pattern count n, the transformation of the whole node T_{cell}, the pattern repetition transformation T_{rep}, and the definition transformation T_{def} specifies the local coordinate system of an instance node. Then it is understood that n instances beginning at an offset of T_{cell} and successively positioned by $T_{rep} \cdot T_{def}$ are defined at that node. That is, the local transformation of instance k of a pattern node is defined as $T_{local.k} = T_{cell} \cdot \prod_{i=1}^{k-1} T_{rep} \cdot T_{def}$. The pattern node also allows back links; that is, the pattern node can point to an ancestor node or itself as its leader. In this way, the approach can support recursive patterns. We present a scene graph representation with a similar interpretation to this one as described in Chapter 5.

4.2.2 Procedural approaches

The basic principle behind procedural approaches is to use algorithms and rules to describe or, in actuality, to generate components. Hence, the types of procedural modeling approaches are diverse, including approaches such as shape grammars, L-systems, fractals, and modeling languages in general.

The variety and spectrum of procedural approaches can be described as a soup of techniques and languages. Rules, operations, commands of various levels of abstraction may be applied or executed with various means of control, ranging from strict convention

to full programmability. To make sense of it all, we may consider classifying approaches as either grammar-based, language-based, or scripting approaches.

4.2.2.1 Grammar-based approaches

We begin by discussing grammar-based approaches. There are two main grammar-based modeling approaches: L-systems and shape grammars. L-systems, originally introduced by Lindenmayer [1968], and Shape grammars, originally introduced by Stiny and Gips [1971], have been used to model organic entities such as trees and plants [Smith 1984; Prusinkiewicz and Lindenmayer 1990] and non-organic entities including cities [Parish and Mueller 2001], buildings [Mueller et al. 2006], facades [Wonka et al. 2003], and even artistic patterns [Cenani and Cagdas 2007]. Both are similar in that they use formal grammars to specify growth or development rules which are used to describe a model. They are different in the way models are grown or developed and in their geometric interpretations.

4.2.2.2 L-systems

In its most basic form, an L-system can be defined as a deterministic context-free grammar defined as a tuple G = {*V*, *S*, ω, *P*}, where V is a set of replaceable symbols (variables), S is a set of terminal symbols (constants), ω is the starting string composed of symbols from V, and P is a set of production rules. Starting from ω, the rules are applied iteratively, where as many rules as possible are applied simultaneously. Hence, L-systems are parallel rewriting systems [Prusinkiewicz and Lindenmayer 1990]. This characteristic makes them appropriate for modeling organic entities which grow.

In order to model shape and geometry, an L-system needs a geometric interpretation. Such geometric interpretations can vary immensely, being able to model anything from the growth of algae to higher plants and trees and even to space filling curves and fractals. For instance, an L-system can be used as an iterated function system to generate fractals.

The most common geometric interpretation is the turtle interpretation, which defines the state of a turtle as a triplet (x, y, α), where x and y define the position and α defines the heading, the direction the turtle is facing. Given a step size and an angle increment, the turtle can respond to the symbols to move, move without drawing, or turn. The turtle interpretation can also be extended to 3D [Abelson and diSessa 1982; Prusinkiewicz and Lindenmayer 1990]. Such a simple geometric interpretation combined with a grammar is able to express a diversity of shapes and geometric designs. For example, a Sierpinski triangle can be drawn using L-system G = {{A, B}, {+, −}, A, {(A → B−A−B), (B → A+B+A)}} with turtle interpretation where both A and B mean "draw forward", + means "turn left by θ", and − means "turn right by θ", where θ=60° (Figure 4.1). The basic L-system can be extended with randomization to model variation (stochastic L-systems). A context-sensitive L-system can be used to simulate interaction between model parts, e.g. the flow of nutrients or hormones in a plant, by using context to pass information. L-systems can also be augmented with positional information to support interactive editing [Onishi et al. 2003; Power et al. 1999] and control shape and other spatial characteristics of growth [Prusinkiewicz et al. 1994; Prusinkiewicz et al. 2001]. Other continuous or more complex phenomena in general can be supported by using some extension of parametric L-systems. For instance, a stochastic, parametric L-system can be used in the procedural modeling of a city [Parish and Mueller 2001].

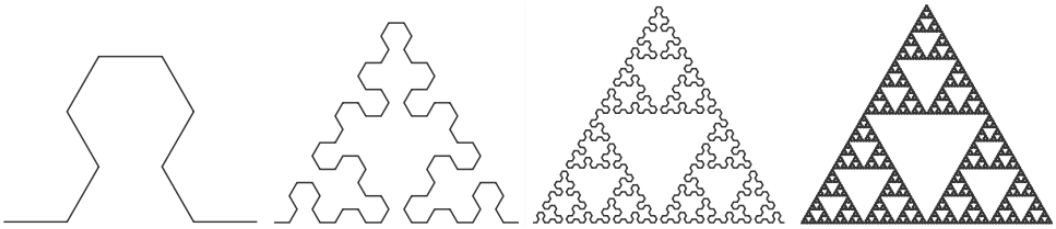

Figure 4.1: Iterations n=2, 4, 6, and 8 of a Sierpinski triangle L-system with turtle interpretation.

4.2.2.3 Shape grammars

The shape grammar of Stiny and Gips [1971] can be defined as a 4-tuple SG = {V_T, V_M, R, I}, where V_T is a finite set of terminal shapes, V_M is a finite shape of non-terminal shapes (called markers), R is a finite set of replacement rules, and I is the initial shape. Shapes are produced by beginning with the initial shape and recursively applying the replacement rules.

Shape grammars generally have a built-in geometric interpretation. The geometric relationships (relative transformations) between the components being replaced and the components being added are part of the replacement rule.

Comparison. The elements of the L-system 4-tuple *V*, *S*, ω, and *P* correspond with the elements of the shape grammar 4-tuple V_M, V_T, I, and R, respectively. One important difference between shape grammars and L-systems is the application of the rules. L-systems apply rules in parallel and simultaneously while shape grammars generally apply them sequentially. Consequently, there are differences in the kinds of models they can express, an enumeration of which is beyond the scope of this review. The important point is that both are sufficiently expressive to model the same recursive patterns that a scene graph can express. In particular, scene graph implementations exist which support

patterns [van Emmerik et al. 1993] and which can be modeled as a deterministic, context-free grammar [Ebert et al. 2003]. Hence, we know that the more expressive grammars (e.g. parametric and context-sensitive) and languages can also support them in general.

4.2.2.4 Language-based approaches

Modeling languages such as Solid Modeling Language (SML) [van Wijk 1986], AML/X [Nackman et al. 1986], and Generative Modeling Language (GML) [GML] support the procedural modeling of shape. The basic idea is use sequences of operations to describe a model [Rossignac et al. 1988; Snyder and Kajiya 1992]. For instance, GML is essentially a programming language for describing shapes. The idea is to represent models as a list of operations which produce an object rather than representing the final result of executing the operations. SML and GML were designed to be able to procedurally process and generate individual shapes and were not intended for specifying and representing coherent arrangements of shapes.

ABCSG [van Emmerik et al. 1993] uses a simple modeling language as the basis for its hypertext approach. A scene graph structure is derived from the language, thus the approach is actually a scene graph approach with a modeling language front-end as the interface. It is a simple approach where its graph supports patterns explicitly and the modeling language conveys the notion of a pattern in a straightforward manner. Hence, borrow the essence of their approach for our representation.

A scene description language such as Scene Description Language (SDL) for the POV-Ray ray tracer [POV-Ray] allows the use of programming constructs to describe and ultimately generate scene content. A scene graph is created in order to facilitate efficient rendering via ray tracing.

4.2.2.5 Scripting approaches

Some modeling software packages include scripting languages which can be used to procedurally execute the modeling functionality of the program. For instance, Maya Embedded Language (MEL) [Maya], Rhinoceros 3D (Rhino) [Rhino3D], and Generative Components [GenComp] scripts can be used to procedurally generate patterns. However, these scripts are not a representation for patterns per se, but can procedurally generate pattern content. When changes are made to the patterns, the scripts ultimately need to be re-executed to re-generate the content.

One advantage of scripting is that it enables the user to flexibly express arbitrary modeling concepts. However, this also has a downside. In general, procedural approaches require skill for programming or rule design. If we know that we want to model patterns, then ideally we would like an integrated modeling approach which distills the essence of patterns and exposes only the relevant parameters to the user. Instead, some procedural approaches are so general that they are essentially trading off relevance for expressiveness beyond what we need.

4.2.3 Summary

We have summarized previously disseminated approaches related to explicitly representing arrangements of patterns. Graph-based approaches, falling under the general category of scene graphs, can explicitly represent arrangements of patterns using a node interpretation specifically designed for patterns. More complex arrangements of patterns are obtained using more general grouping nodes. Recursive structures can be supported using back links. Procedural approaches rely on interpreting, applying, or executing a list of rules, operations, or commands. Because they are general, with some approaches using

modern language constructs, they are sufficiently expressive to model hierarchical and recursive pattern structures. Hierarchical and recursive semantic relationships can be captured, but are not stored in a way that supports incremental updates based on change in pattern parameters and attributes. Instead, the procedure (or "history") is re-executed in order for the model to be re-evaluated. Furthermore, procedural approaches require skill for programming or rule design. Ideally, a pattern modeling approach should distill the essence of patterns and expose only the relevant parameters to the user. Instead, some procedural approaches are so general that they trade off relevance for expressiveness beyond what we require.

4.3 Persistence (naming)

After a model is created, certain components and features may be identified for further editing. Attributes or constraints may be assigned to or new components may be added relative to the identified component. The challenge arises when the underlying model is modified. To maintain the assigned attributes, constraints, or child components, the need arises to continue to identify or persistently *name* the same components.

This persistent naming problem for parametric, feature, and history based modeling has been well studied. Marcheix and Pierra [2002] present a survey on existing approaches.

Fortunately, our version of the problem is less complicated. We directly name the occurrences or instances and their existence or location in the hierarchy is explicit. Compare this with the persistent naming problem, where the structure of topological features may be implicit depending on where and how they are combined. Hence, for our

approach we need only describe how to explicitly and uniquely identify each instance in an assembly.

5 PATTERN REPRESENTATION

Here we propose a representation for regular patterns. The representation is based on work by van Emmerik et al. [1993] and supports simple patterns, nested patterns, grouped patterns, and recursive patterns. Thus it is powerful enough to describe a wide variety of configurations of a variety of geometric entities while at the same time being concise and compact.

5.1 Graph Representation

We now describe a graph representation for regular patterns.

Consider a regular pattern P = (F_1, c, T) defined using three components: a pattern leader F_1, a pattern transform T, and a pattern count c. These three components can be represented in a rooted directed graph with two kinds of nodes: leaf (terminal) nodes and interior nodes. All leaf nodes, which we call *geometry nodes*, represent shapes. All interior nodes represent patterns and so we call them *pattern nodes*.

This rooted directed graph is a binary tree, but with the added characteristic that it is allowed to link back on itself. That is, a node can link to an ancestor node, or a sibling/relative node, or itself as a child node. Since the graph is essentially a binary tree, each pattern node (interior node) can have a left child node and a right child node.

Given a pattern node n (Figure 5.1), the pattern count c is stored as n.o. The node n.L linked to by the left link (pattern link) is the pattern leader of the pattern defined at node n. The pattern transform T is associated with the left link and is stored as n.l. Thus, the node

n with its left link and left child node encodes a pattern (n.L, n.o, n.l). Figure 5.3 gives an example.

While the left link n.l and left child node n.L of node n are used to encode the pattern semantic, the right link n.r and the right child node n.R are used to encode the grouping semantic. The pattern defined at the right child node n.R is grouped to the pattern defined at node n. The right link n.r stores the relative transformation between the frame of the pattern at n and the frame of the pattern at n.R. The pattern defined at right child node n.R is said to be *grouped* to the pattern defined at node n. Figure 5.6 gives an example.

Recursive patterns are supported by maintaining a recursion count for the left and right child of each node. We use n.cl and n.cr to denote these counts corresponding to n.L and n.R respectively.

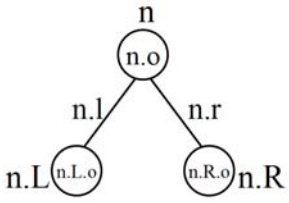

Figure 5.1: Notation for a pattern node n.

In our pattern representation, pattern nodes must reference a left child node since all patterns have a pattern leader. Furthermore, a geometry node may only exist as a left child node of a pattern node; that is, a geometry node cannot be a right child node. Hence, all geometric components referenced in our graph notation must belong to a pattern. A component that occurs only once would be represented as a pattern of one instance. Patterns may be nested by linking a pattern node as a left child node and patterns may be grouped by linking a pattern node as a right child node. In this way, a complex scene or

assembly of geometric components can be described by combining simple patterns. Designating the right link to be the grouping link implies that pattern nodes are not required to reference a right child node since grouping an additional pattern to a given pattern is optional.

5.1.1 Simple Patterns

We now give an example of a complex pattern built from simple patterns using grouping and nesting.

We begin with a simple pattern, the most basic unit for building a model. According to our graph representation, a single isolated geometric entity C is represented as a pattern of one C. The pattern count of the pattern node is one and its left child node contains C. A pattern of one instance does not use the pattern transform. We can use an expression P_{1CHAIR}=1C (or simply P_{1CHAIR}=C) to describe this graph (Figure 5.2). In this notation, the coefficient to the patterned component specifies the pattern count and implies the existence of a pattern transform. Note that this pattern transform is defined in a separate way as discussed later in this thesis.

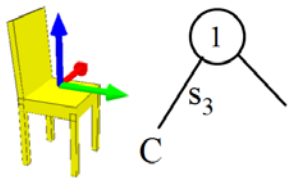

Figure 5.2: Single instance pattern (C, 1, s_3) is specified as P_{1CHAIR}=C. The pattern transform s3, which specifies the transformation between successive instances in a pattern, is not used in a single instance pattern.

Multiple instances can be defined by increasing the pattern count and defining the pattern transform. Given the frame of the first instance, the frames for successive instances are computed by successively applying the pattern transform to the first frame. For instance, to compute the frame of the fourth chair in the pattern we apply $s_3s_3s_3$ to the frame of the first chair, where s_3 is the pattern transform. In other words, the relative transformation between the pose of the first chair and the pose of the fourth chair is $s_3s_3s_3$. More formally, the local frame for instance k of a pattern is defined as $T_{local.k} = (T_{rep})^{k-1}$ or:

$$T_{local.k} = \prod_{i=1}^{k-1} T_{rep} ,$$

where T_{rep} is the pattern transform. We can use the expression $P_{5CHAIRS}$=5C to describe a graph representing a pattern of five chairs (Figure 5.3).

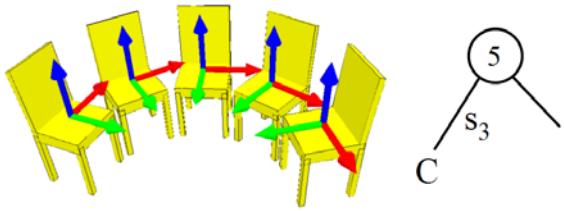

Figure 5.3: Multiple instance pattern (C, 5, s_3) is specified as $P_{5CHAIRS}$=5C.

5.1.2 Nested Patterns

When a pattern is used as the leader of another pattern, this forms a pattern of a pattern. In our graph notation, this means that a pattern node has a left link to another pattern node. We can use the two expressions $P_{4x5CHAIRS}$=4F and F=5C together to describe a graph representing a pattern of four rows each of which are a pattern of five chairs. (Figure 5.4). (Note that we have the freedom to rename F=$P_{5CHAIRS}$.)

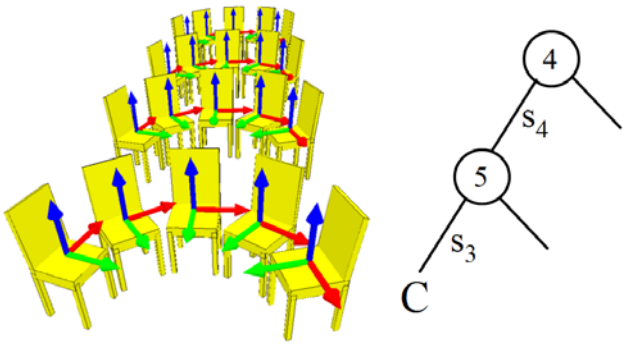

Figure 5.4: Pattern (F, 4, s_4) of a pattern F = (C, 5, s_3) is specified as $P_{4x5CHAIRS}$=4F and F=5C. The resulting graph can be written as ((C, 5, s_3), 4, s_4).

In this example, the local frame of the k-th chair in the j-th row can be computed as:

$$T_{local.j.k} = \prod_{i=1}^{j-1} s_4 \cdot \prod_{i=1}^{k-1} s_3 \ .$$

5.1.3 Grouped Patterns

Multiple patterns can be *grouped* to form a compound geometric component. This includes grouping multiple instance patterns and single isolated components, which are considered single instance patterns.

A pattern is grouped to another by hanging it off the right link (grouping link). The transform associated with that link specifies the relative transformation between the two patterns being grouped by that link. The nesting relationship determines which pattern is relative to the other. The child pattern has a frame that is relative to the parent pattern. We can use the expression D=T+C (or D=1T+1C) to describe a graph representing the grouping of a pattern of one table with a pattern of one chair, i.e. a table grouped to a chair (Figure 5.5). The ordering of the terms in the expression determine the nesting

relationship. The patterns described by terms to the right are nested under the patterns described by terms to the left. The grouping symbol "+" asserts that the term to its right is nested as a right child under the term to its left. The grouping symbol also implies the existence of a relative transformation. Again, we will discuss how to specify this transformation later in the thesis.

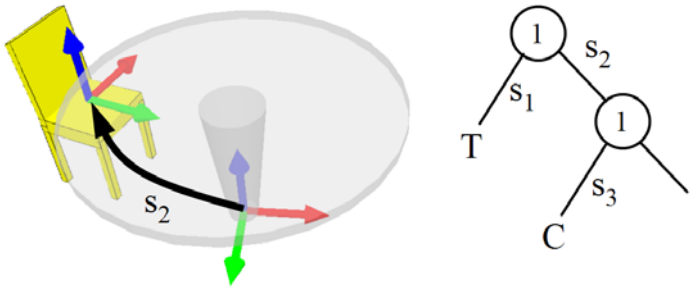

Figure 5.5: Grouping two single instance patterns D=T+C.

Increasing the coefficient of C increases the pattern count for the pattern of chairs. For example, the expression S=T+5C describes a pattern of one table T grouped to a pattern of five chairs 5C to form one dining set entity S (Figure 5.6). We can obtain the frame for any component in the graph relative to the root by traversing the graph from the root and accumulating transformations. (This accumulation is accomplished by multiplying homogeneous transformation matrices.) For example, the frame of the fourth chair in the dining set S has a relative transformation $s_2s_3s_3s_3$ to the frame of the (first and only) table, which is the local frame at the root of the graph for S.

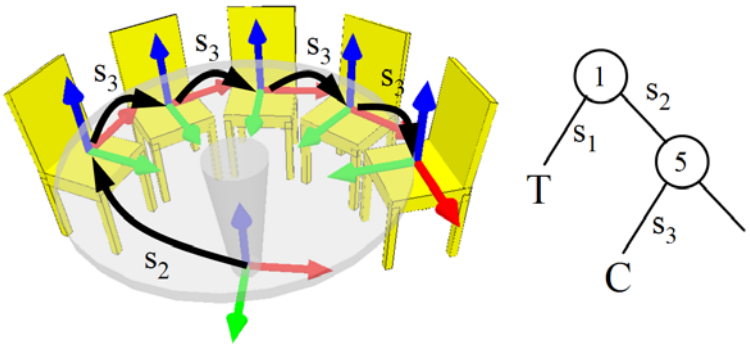

Figure 5.6: Grouping a single instance pattern with a multiple instance pattern S=T+5C.

A more complex component (Figure 5.7 left) could for example be defined by D_2=T+5C+6C. In this example, a pattern of five chairs is grouped with a pattern of one table. Then, a pattern of six chairs is grouped to the pattern of five chairs (and are consequently also grouped to the pattern of one table). Again, we can find the relative transformation between the root and any component by accumulating transformations while traversing the graph. For example, if we let d_4 be a translation which defines a second chair behind the first one and let d_5 be a rotation around the center of T, but by a slightly smaller angle than s_3, we can obtain the frame of the fifth chair on the back row by applying the transform $s_2d_4d_5d_5d_5d_5$ to the local frame at the root.

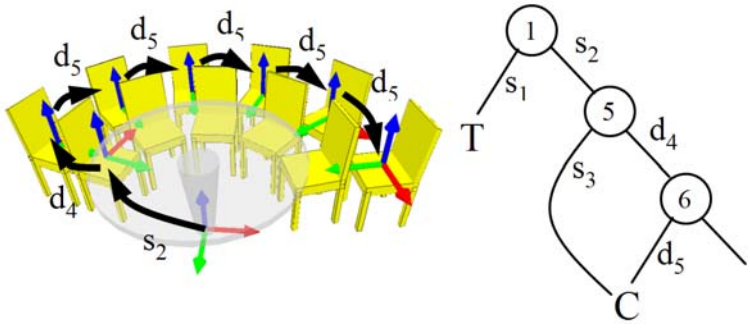

Figure 5.7: Grouping additional patterns in a chain D2=T+5C+6C.

Note that the corresponding graph (Figure 5.7 right) is not a tree, since two components reference C. (It is a rooted directed acyclic graph (RDAG).) However, the traversal of the graph for instantiating its components is not affected. We can show this by replicating the shared link. This results in a graph which has an equivalent depth-first traversal. We further discuss this equivalence in Section 5.2.

5.1.4 Recursion of Patterns

When a pattern itself is referenced or contained in its own pattern leader, this forms a recursive pattern. This occurs when a pattern-leader contains a reference to an ancestor component. Since expanding the ancestor node involves revisiting the same node, scene components in the revisited sub-graph are recursively defined. Consequently, a scene graph containing a recursive pattern is a directed graph with one or more cycles.

A simple example of a recursive pattern is given in Figure 5.8, where the expression $M=C+3M^5$ specifies a cylinder plus a pattern of three more cylinders recursively to a depth of 5. In this case, the user specifies a recursive pattern by referencing the component M itself in its definition. In general, whenever a component is defined using a component that ultimately references itself in its sub-hierarchy (through nesting or grouping), a recursive pattern is created. Since this results in an infinite component, the user also specifies a recursion limit. This recursion limit is specified as a superscript of the recursively referenced symbol in the expression (as in the digit `5' in "$M=C+3M^5$"). When a recursive term is evaluated, its own superscript is decremented. For example, when evaluating the term "$3M^5$", the symbol M evaluates to $C+3M^4$; hence, $C+3M^5$ becomes $C+3(C+3M^4)$. It is important to note that this differs from the standard algebraic interpretation for which $M \neq C + 3M^5$ unless both C and M are null. In our usage, the

superscript is a parameter and not an exponential. Hence, we could write the expression as M(r) = C + 3M(r-1), where M(0) ≡ null.

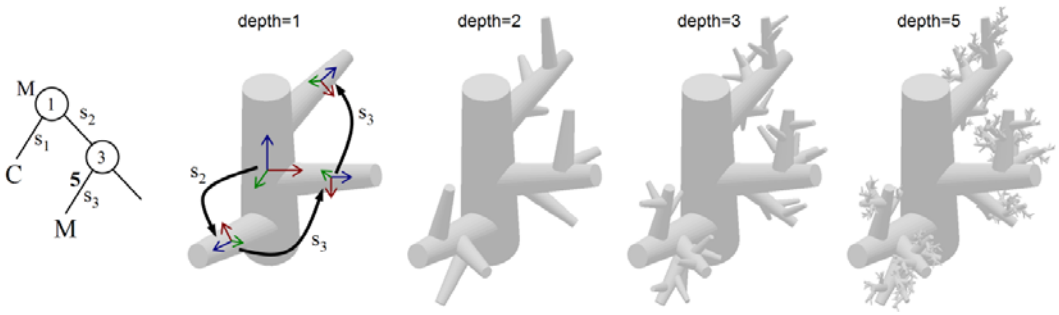

Figure 5.8: Recursive trees defined by $M=C+3M^1$, $M=C+3M^2$, $M=C+3M^3$, and $M=C+3M^5$ respectively. The graph for $M=C+3M^5$ is given (left).

Consider a second example where grouping is used to create multiple tree branches with different recursion depths. In the expression $M=C+M^3+M^2$, an ambiguity apparently occurs during the evaluation of the expression. For instance, in the initial evaluation of M^3, the expression becomes $C+M^2+M^2$. Now if we expand the left branch (which evaluates to $C+M^1+M^2$) and then the right, we get $C+M^1+M^1$. If we expand the other way around, i.e. the right branch (which evaluates to $C+M^2+M^1$) and then the left, we will also get $C+M^1+M^1$. However, since each refer to different branches and thus are on unique paths, there is no ambiguity. The visualization in Figure 5.9 illustrates the uniqueness of each branch and the state of the two recursion depths corresponding to the branch. Even though the state of the two recursion depths is not unique, the branches themselves are unique since their path is unique. For example, state 2,1 ($C+M^2+M^1$) occurs twice but clearly on two different branches. Even if the sequence of transformations evaluated results in the same frame for multiple instances, they are still separate, yet coinciding, instances.

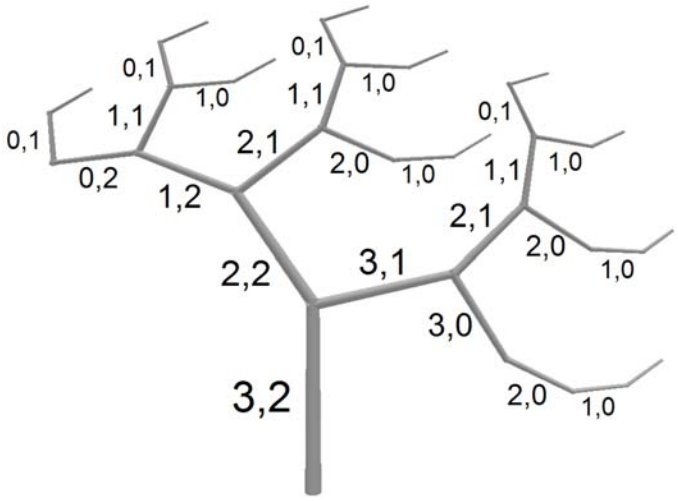

Figure 5.9: A tree defined by $M=C+M^3+M^2$ has multiple differing recursion depths. Each branch in the tree is uniquely defined by a unique evaluation path even though the state of the two recursion depth parameters may not be unique. For example, all of the end branches (not labeled) have the same terminating state 0,0.

To clarify the interpretation of the recursion counts, we may describe the assembly $M=C+M^a+M^b$ parametrically as $M(a, b) = C + M(a-1, b) + M(a, b-1)$, where $M(a, b)\equiv$null if $a|b<0$. Hence,

$M(0, b) = C + \text{null} + M(0, b-1) = C + M(0, b-1)$
$M(a, 0) = C + M(a-1, 0) + \text{null} = C + M(a-1, 0)$
$M(0, 0) = C + \text{null} + \text{null} = C$

We can extend this idea to handle additional separate recursive terms $M = C + M^a + M^b + M^c + M^d + ...$, for example $M = C + M^3 + M^2 + M^3 + M^3 + \ldots$. Then, to describe the evaluation of the recursion we may write the parametric expression $M(a,b,c,d,...) = C + M(a-1,b,c,d) + M(a,b-1,c,d) + M(a,b,c-1,d) + M(a,b,c,d-1) + \ldots$ (since each recursive call only decrements its own recursion counter) and we define $M(a,b,c,d,...)\equiv$null if $(a|b|c|d|...<0)$. While such expressions may become cumbersome to write and expand in

text form, based on our graph representation we can design a simple traversal algorithm which can evaluate such models.

We have described a graph representation for regular patterns that supports simple, grouped, nested, and recursive patterns. We next describe a traversal algorithm for rendering the model represented by a pattern graph and accessing its instances.

5.2 Traversal algorithm

To render an assembly model or to access each instance and its final frame, we use a recursive depth-first traversal of the graph. The traversal first processes the pattern at the left node by saving the current accumulated transformation (pushMatrix()), evaluating the pattern in a loop, and then restoring the accumulated transformation (popMatrix()). Evaluating the pattern in a loop involves a recursive call to evaluate the instance represented by leader n.L, applying the pattern transform n.l, and then continuing to loop through the rest of the instances. The right node is then processed by saving the current accumulated transformation, applying the grouping transform (relative transformation between the two patterns), making a recursive call to evaluate the grouped pattern, and then restoring the accumulated transformation. This traversal is illustrated by the procedure eval(n), where n is a node in the graph.

```
eval(n) {
  if (isPrimitive(n)) process(n);
  else {
    pushMatrix();
    for (int i=1; i<=n.o; i++) {
      eval(n.L);
      applyMatrix(n.l);
    }
    popMatrix();
    pushMatrix();
    applyMatrix(n.r);
    eval(n.R);
    popMatrix();
  }
```

```
}
```

Essentially the same instantiation algorithm may be used for both recursive and non-recursive pattern hierarchies. The only addition to the basic algorithm needed for supporting recursive patterns is to check the recursion limit, decrement it when making a recursive call, and increment it when returning (i.e. the decrement and increment of n.cl and n.cr occur where the pushMatrix() and popMatrix() calls are made for the left and right nodes, respectively.) The procedure evalr(n) incorporates recursion limits to the basic traversal.

```
evalr(n) {
  if (isPrimitive(n)) process(n);
  else {
    if (n.cl > 0) {
      n.cl--;
      pushMatrix();
      for (int i=1; i<=n.o; i++) {
        eval(n.L);
        applyMatrix(n.l);
      }
      popMatrix();
      n.cl++ ;
    }
    if (n.cr > 0) {
      n.cr--;
      pushMatrix();
      applyMatrix(n.r);
      eval(n.R);
      popMatrix();
      n.cr++ ;
    }
  }
}
```

As long as a finite recursion limit is specified for every back-reference (recursive call) in the graph, the traversal is finite and thus always terminate. Also note that in the case where multiple pattern nodes reference the same subgraph, the traversal is also finite since the subgraphs are finite. We can show this by producing an equivalent expanded tree without the back-references by replicating the nodes that are referenced multiple

times (Figure 5.10). Figure 5.9 also provides an example of terminating recursion (for the case of multiple branching recursion).

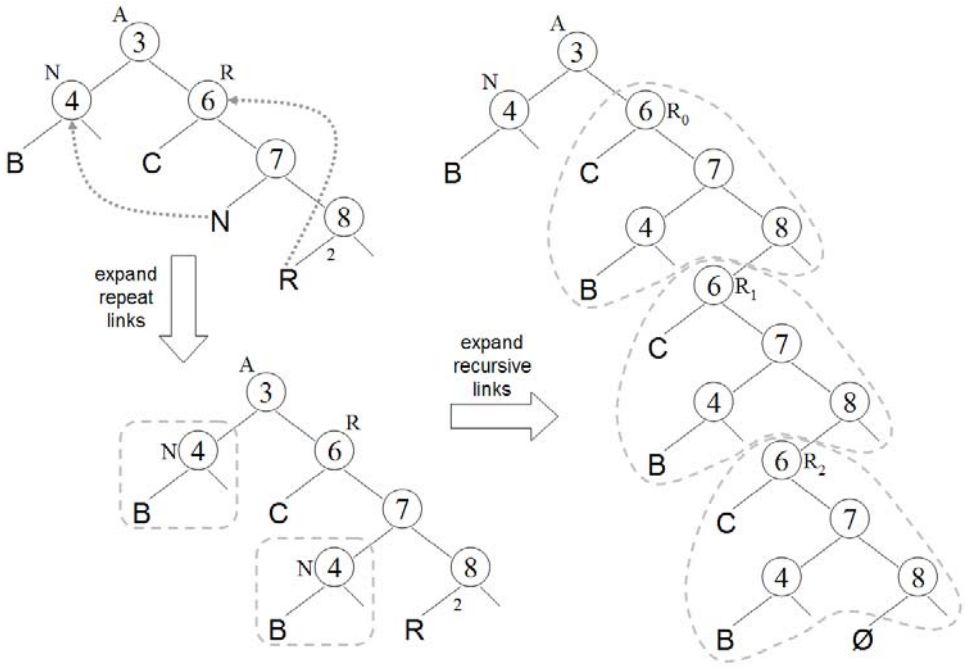

Figure 5.10: The expressions A = 3N + 6C + 7N + 8R^2 and N = 4B result in a directed graph with nodes N and R that are referenced multiple times. The graph can be expanded into an equivalent tree since a finite recursion limit is specified for the recursive reference to R.

5.3 Interpretation/semantic of traversal

Instance as a path. An instance is uniquely identified by a *path* from the root to a leaf in the rooted directed graph. This path includes looping through pattern links. Each different path from the root to a leaf follows a different sequence of transformations. Even in the case where the resulting accumulated transformations from two different paths are the same, they still represent two distinct instances which happen to be coincident. Example paths are shown in Figure 6.3 and Figure 6.4.

Instance defined not just at leaves (compound features defined at interior nodes). A compound feature (including patterns and assemblies) is defined at each interior node. The path to that node identifies that compound component. Hence, instances are defined not only at the leaf nodes but also at all interior nodes. We can also say that each interior node (including the root) is semantically equivalent to an instance; evaluating the subgraph of a node determines the contents of the instance. An example is given in Figure 5.18.

Frames for each instance being an accumulation of transforms. To determine the frame at any node in the graph, traverse the graph from the root while accumulating transformations encountered across the links visited. This process includes looping through the pattern links. Hence, the transformations accumulated on the path which identify an instance will define the frame for the instance relative to the frame of the root. Example paths are given in Figure 5.5, Figure 5.6, Figure 5.7, and Figure 5.8.

An assembly not being instantiated without a frame. We have defined an instance as a feature transformed by a frame. The root of an assembly graph represents a compound feature but does not have a frame associated with it. Thus it is necessary to instantiate the whole graph at the root and thus instantiate all of its constituent components. This is to "seed" the assembly graph. In the graph, all transformations are relative to the parent. Since a frame is not defined at the root, an assembly graph represents an uninstantiated feature and is semantically equivalent to a shape. It needs to be positioned using a frame in order to instantiate it. This instantiation at the root propagates the transformation down the tree. The frame for a child node is defined relative to its parent using the grouping transform at the right link of its parent. This

includes parents by recursion. The frame for an instance in a pattern is defined relative to its predecessor in the pattern sequence using the patterning transform.

5.4 Pattern editing

In this section we summarize the ways a pattern described with our representation can be edited. (To be clear, we do not claim any contributions in this section, as it primarily summarizes work presented by van Emmerik et al. [1993], but we include it for background with respect to our contribution. Note that the main contribution of Part I of this thesis is related to selecting subsets of primitives in an assembly graph, which we describe in detail in Chapter 6. Hence, the reader may safely skip this section and use it for further reference, in particular when we refer to it in Section 6.5.1.) Now in particular, the design and editing of patterns is achieved by specifying an assembly graph and adjusting its parameters. The assembly graph defines which and how many items are included in the assembly and how they are related semantically, i.e. how they are grouped or nested. The inventory of items included is indicated by leaf nodes, their counts in the interior pattern nodes, and their semantic relation by whether they are nested to the left or right of their parent node. Transformations embedded at the graph links define how the items are positioned and oriented with respect to other items with which they have semantic relations including grouping, nesting, and patterning.

For a designer to specify and edit a graph in practice, a modeling system needs to provide a set of editing operations to build its structure and adjust its parameters, including transformations. However, the particular presentation of these operations to the designer is application dependent and may or may not directly reflect or expose the graph representation. Hence, instead of providing a detailed discussion of how a designer can

create and edit an assembly graph, we focus our discussion on enumerating some of the most relevant possible changes to the graph, its parameters, and the transformations embedded in it. We then give suggestions on what kind of editing operations might invoke these changes, but will not go into detail on exactly how the designer would invoke them. For a detailed discussion of a possible user interface by which the user can define and edit an assembly graph, we refer the interested reader to [van Emmerik et al. 1993].

5.4.1 Notation for describing assembly graphs

In our discussion of our assembly graph representation, we used a basic notation, e.g. $A = 7B + 12C + 3A^4$, as a text-based way to describe a graph. In this chapter, we use it to help us explain the possible graph editing operations. Hence, we begin by providing additional details on the notation here.

In our notation, a pattern assembly graph can be described using m expressions $A_i = n_{i1}F_{i1}$ [+ $n_{i2}F_{i2}$ [+ $n_{i3}F_{i3}$ […n_{in},F_{in}]]] of n terms, where i = 1, 2, 3, …m. The left hand side (LHS) of the expression is a symbol A_i representing an assembly, which is a feature. Each right hand side (RHS) term $n_{ij}F_{ij}$, which we call a *pattern term*, represents a pattern consisting of pattern count n_{ij} and feature F_{ij}. Each pattern is grouped using a grouping operator (e.g. "+"). Each feature F is just a placeholder for a symbol that represents a shape or a LHS symbol A_i. We use single letter symbols for feature names. Hence, the notation describes an assembly as a list of patterns, a pattern as a feature with a count, and a feature as either a shape or an assembly therein. We also designate a single root LHS symbol A_1 which can be considered the start or seed symbol representing the whole

assembly. In general, any LHS symbol is the seed symbol for some sub-assembly of the whole.

In the case of a recursive reference made in a RHS pattern term, recursion depth limit is indicated by an exponent to the symbol in the pattern term. For convenience, we write this exponent as a coefficient to the right of the symbol. For example, $M=C+3M^5$ can be written as M=C+3M5 to describe a recursive pattern with a recursion limit of 5. Placing the recursion limit coefficient to the right differentiates it from the pattern count coefficient, which is written to the left of the symbol.

Other grouping operators besides "+" (superposition) can be used. For instance, the assembly (union for solids) operator ("+") could be replaced with other operators such as difference ("–") or intersection ("*"), which are operators commonly used in CSG. Figure 5.11 provides examples. Regardless of what operator is used, each results in logical grouping even though their geometric effect may be different.

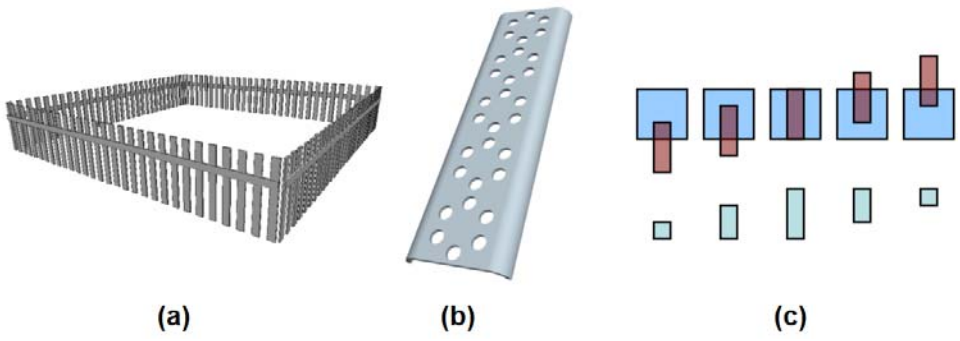

Figure 5.11: Examples designed using different grouping operators. (a) A fence is defined as a pattern F=4R of 4 rows, each defined as a combination R=30V+H of a pattern of 30 vertical beams and one horizontal. (b) A CSG model of a fuselage plate is defined as F=P-5C, a plate from which one has subtracted a pattern of five arrangements C, each defined as a pattern C=6H of 6 holes. (c) A simple design D=5A*5B is defined using the intersection operator.

5.4.2 Creating assembly graphs

One way a graph can be constructed is by parsing a set of expressions in our notation. Practically, LHS symbols do not directly correspond to nodes in the graph. LHS symbols are dummy or placeholder names for components in the assembly. Only when RHS terms are encountered are graph nodes defined. For instance, the seed symbol is a placeholder representing the whole graph. The actual root node directly corresponds to the first pattern term on the RHS of the seed expression.

Simple pattern. Parsing a RHS pattern term n_iF_i defines a pattern node with pattern count n_i (Figure 5.12). Its left child node is determined by evaluating the symbol F_i which represents the pattern leader. Its right child node may or may not exist depending on whether there are additional pattern terms in the expression (grouping).

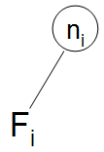

Figure 5.12: Node structure defined by pattern term n_iF_i.

Shapes. Certain symbols may refer to shapes. One possible way to indicate this is to list them separately along with a description of the shape. Then when parsing, the list is checked to see if it is a shape. If it is listed as a shape, the node type can be flagged as a leaf node and the appropriate shape information linked. If it is not, there should exist another expression to parse with the symbol on the LHS which defines it.

Nesting. Unless it refers to a shape, every RHS symbol defines nesting to the left. When evaluating a RHS pattern term n_pF_p, the pattern leader F_p is determined by parsing

an expression with LHS symbol F_p. Given such an expression $F_p=n_cF_c+n_iF_i+\ldots$, the first RHS pattern term n_cF_c corresponds to a node which is nested as the left child node of the pattern node corresponding to n_pF_p, which is its parent. For example, $A_x = n_pA_p$, $A_p = n_cF_c$ directly nests a pattern of n_c instances of F_c inside a pattern of n_p instances resulting in a doubly nested pattern of $n_p \times n_c$ instances of F_c (Figure 5.13).

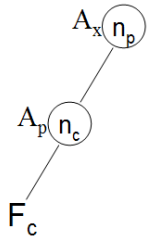

Figure 5.13: Node structure defined by $A_x = n_pA_p$, $A_p = n_cF_c$.

Grouping. The presence of a grouping operator (e.g. "+") in an expression indicates that the node represented by the pattern term immediately to the right of the operator is to be nested as a right child of the node represented by the pattern term immediately to the left of the operator. Hence, if n_iF_i is not the first term in the expression, the pattern represented by the term is a right child node of the pattern represented by the term $n_{i-1}F_{i-1}$ which immediately precedes it in the expression, e.g. $A=n_{i-1}F_{i-1}+n_iF_i$ (Figure 5.14). Thus we see that patterns are grouped to the right and nested to the left.

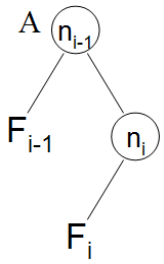

Figure 5.14: Node structure defined by $A=n_{i-1}F_{i-1}+n_iF_i$.

Recursion. One way to define a recursive link in the assembly graph is to specify a RHS symbol, say X, which is the LHS symbol in the same expression, e.g. X=W+3X. To specify a finite recursive structure, a recursion limit should be specified, e.g. X=W+3X4 specifies a recursion limit of 4 for the recursive evaluation of X in the pattern term 3X.

A more complex recursive structure is formed when the RHS symbol, say Y, indirectly references the LHS symbol which uses Y in its definition. For example, in the expressions Y=W+5X, X=4Z+3V, and Z=V+3Y4, evaluating Y ultimately requires evaluating Z=V+3Y4 which uses Y in its definition. Hence, a recursive reference back to Y=W+5X is needed (Figure 5.15). Due to the recursion limit 4 in the pattern term 3Y4, this recursive reference is allowed evaluate deeper 4 times (though the total number of evaluations is 3^4 times.

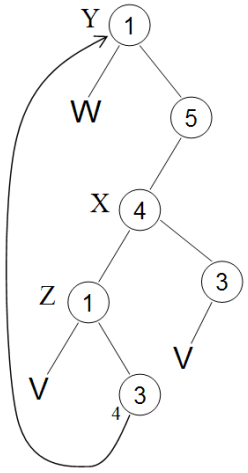

Figure 5.15: Node structure defined by Y=W+5X, X=4Z+3V, and Z=V+3Y4.

With respect to parsing, any back referenced symbol should have already been visited by the parser. Hence, whenever a symbol is evaluated, the parser only needs to check whether the symbol has already been visited. Visited symbols can be stored in a list. If the symbol is found in the list, the parser knows which node to link as its left child node and also to make sure there is a recursion limit indicated in the pattern term.

Validity. A collection of expressions represents a complete and valid assembly graph under the following two conditions: (1) All RHS symbols are defined, that is, they correspond to a LHS symbols or are defined as shapes. (2) Each LHS symbol is uniquely defined using one expression. Note that unreferenced LHS symbols can be safely ignored.

5.4.3 Editing assembly graphs

We now enumerate the ways to edit a pattern assembly graph and its parameters. To summarize, we can change the pattern count and recursion limit, delete/insert a right or left child node, delete a leaf node (special case of deleting left child node), delete a subtree, and edit the grouping and pattern transforms. To support these editing operations,

we first need to provide the designer a way to pick nodes in a graph. This can be achieved by selecting nodes directly in an outline view of the graph or by clicking on a textual representation of the graph as described in [van Emmerik et al. 1993].

Change pattern count. The pattern count is a parameter n.o>0 stored at each pattern node. To change the pattern count, the designer needs some way of selecting the node, e.g. using an outline view or textual representation as in S=T+5C → S=T+8C.

Recursion limit. The recursion limit is a parameter n.rc stored at each pattern node. Pattern nodes. It may be set to a dummy value, e.g. -2, or set to 1 by default so that all links (whether they cause recursion or not) are evaluated. The designer may edit a recursion limit in a manner similar to the pattern count. For example, M=C+3M5 → M=C+3M3 depicts a change of the recursion limit from n.rc=5 to n.rc=3 for the recursive evaluation of M.

Delete/insert a right child node. Given a node n and its right child node n.R, when n.R is deleted, the right child node n.R.R of n.R becomes the new right child node of n, i.e. setting n.R=n.R.R. This scenario may occur when, for example, a pattern term is removed from an expression, excluding the first term in the exception, e.g. S=3A+4B+5C → S=3A+5C (Figure 5.16a). Conversely, adding a pattern term to an expression (other than the first term) may be interpreted as inserting a right child node, e.g. S=3A+5C → S=3A+4B+5C (Figure 5.16b). In this case, the graph structure is updated by setting m.R=n.R and n.R=m, where n is the node corresponding to the pattern term immediately preceding the newly added pattern term and m is the new node corresponding to the newly added pattern term.

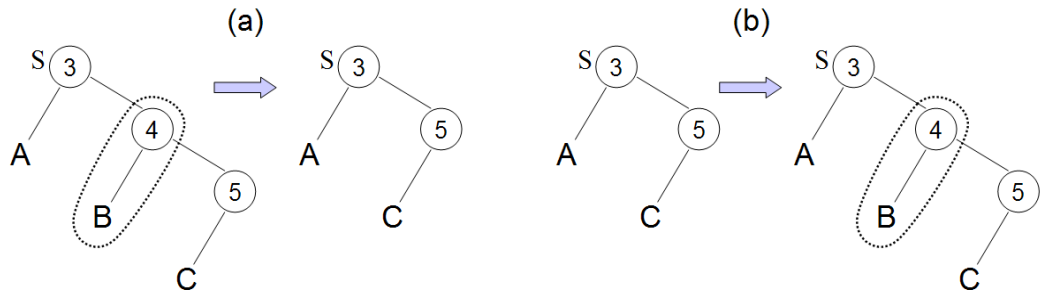

Figure 5.16: (a) A right child node is deleted as a result of removing a pattern term that is not the first in the expression (S=3A+4B+5C → S=3A+5C) . (b) A right child node is inserted as a result of inserting a pattern term not at the beginning of the expression (S=3A+5C → S=3A+4B+5C).

Delete/insert a left child node. Given a node n and its left child node n.L, when n.L is deleted, the right child node n.L.R of n.L becomes the new left child node of n, i.e. setting n.L=n.L.R. This scenario may occur when, for example, the first pattern term of an expression is removed, e.g. R=2S, S=3A+4B+5C → R=2S, S=4B+5C (Figure 5.17a). If the LHS symbol of the expression is the seed symbol, the node represented by the term immediately following the deleted term becomes the new root of the graph. Conversely, adding a pattern term at the beginning of the RHS of an expression may be interpreted as inserting a left child node, e.g. R=2S, S=4B+5C → R=2S, S=3A+4B+5C (Figure 5.17b). In this case, the graph structure is updated by setting m.R=n.L and n.L=m, where n is the node corresponding to the term which refers to the LHS symbol of the expression in which the term corresponding to the new node m is being inserted.

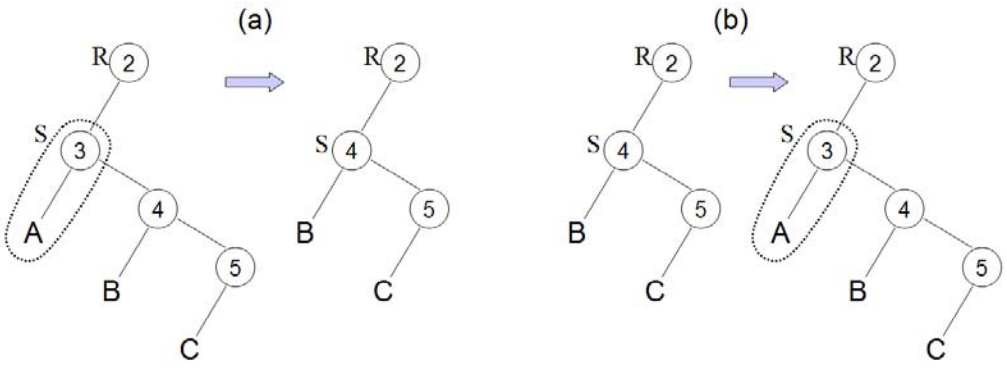

Figure 5.17: (a) A left child node is deleted as a result of removing the first pattern term in a referenced expression (R=2S, S=3A+4B+5C → R=2S, S=4B+5C). (b) A left child node is inserted as a result of inserting the first pattern term in a referenced expression (R=2S, S=4B+5C → R=2S, S=3A+4B+5C).

Delete subtree. Deleting an entire subtree of a graph is valid if the root of the subtree is nested as a right child node. This scenario may occur when the last p pattern terms of an expression are removed. The root of this subtree is the node corresponding to the first of these p pattern terms. If the root of the subtree is nested as a left child node, this invalidates the graph since every interior (pattern) node is required to have a left child node since every pattern is required to have a leader.

Edit transforms. There are two kinds of transformations: the pattern transform and the grouping transform. The pattern transform, indicating the relative transform between successive instances in a pattern, is stored as n.l at the pattern node n which also stores the corresponding pattern count n.o and links to the pattern leader as the left child node n.L. Hence, there is one pattern transform associated with each and every pattern term in an expression in our notation. One possible way of picking a particular pattern transform for editing would be to click on the corresponding pattern term. The grouping transform, indicating the relative transform between successively grouped patterns, is stored as n.r at the pattern node n which also stores the link to the corresponding grouped pattern as the

right child node n.R. There is one grouping transform associated with each and every grouping operator in an expression in our notation; hence, they represent one way of picking a particular grouping transform for editing. For more details on how the designer picks and edits transformations, see [van Emmerik et al. 1993].

This concludes our enumeration of ways to edit a pattern assembly specified in our graph representation. We have not provided an exhaustive list of all possible ways to edit a graph in general, but we have sought to provide a list which includes some of the most relevant edits in the context of computer-aided design and modeling.

5.5 Pattern examples

In this section, we provide additional examples of scenes described using our graph-based assembly of patterns representation. We give four examples: a bar scene, a spiral staircase, a Sierpinski-inspired gasket, and a grove of trees.

5.5.1 Bar scene

Figure 5.18 depicts a bar scene. The expression D = T + 5C can be used to describe a dining set as a pattern of one table grouped with a pattern of five chairs. The expression R = 4D can be used to describe a pattern of four dining sets in a row. Adding the expression F = 3R nests the node describing a row as the left child of a pattern node of three instances. Finally, the expression B = 5F describes a pattern of five floors.

In this example, we described the construction of a scene from the bottom-up, building from simple components up to more complex. The relationships of the subcomponents are retained in the graph. Furthermore, four simple expressions describe a scene with 300 chairs uniformly tucked under 60 tables. Because the graph evaluates the

patterns on the fly and all the subcomponents inherit the transformations, a designer could trivially modify the pattern counts and transforms and see the results immediately. For instance, the designer might update B=5F → B=2F to reduce the amount of floors from five to two (as in Figure 5.18). Just as easily, the designer can increase the number of dining sets in a row and the result would be reflected in every row on every floor.

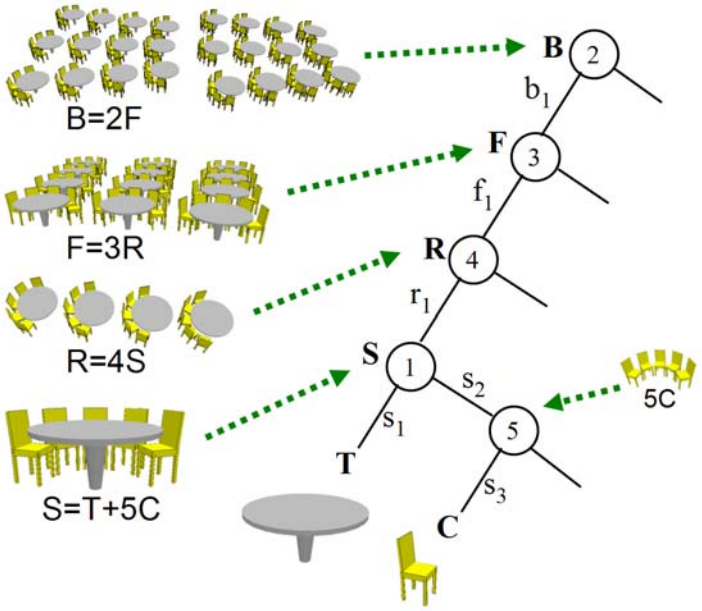

Figure 5.18: A two floor version of the bar scene.

5.5.2 Spiral staircase

Here we describe how a designer might create a spiral staircase with tiled stairs. The designer begins by defining a stair step as a box and then defining a pattern of 26 steps next to a cylindrical column: A = 26S + C, S = B, where "C" and "B" are special symbols referring to a cylinder and a box, respectively. The designer moves the second instance of the step pattern up and lines up the top step with the top of the column (Figure 5.19a).

The designer then moves the second step slightly around the column and the system computes a screw motion (Figure 5.19b). Realizing the steps are too steep, the designer moves the second step further around the column (Figure 5.19c). Now the designer defines a row of tiles as R = 10T and adjusts the pattern transform (Figure 5.19d). Finally, the designer adds a pattern of four rows of tiles to the step definition S = B + 4R, adjusts the grouping transform between the tiles and block by adjusting the first instance, and adjusts the pattern transform between the rows by adjusting the second row (Figure 5.19e). The resulting model has 1040 tile instances, 26 block instances, and 1 cylinder instance represented by a graph with five pattern nodes and three primitive nodes (Figure 5.19f).

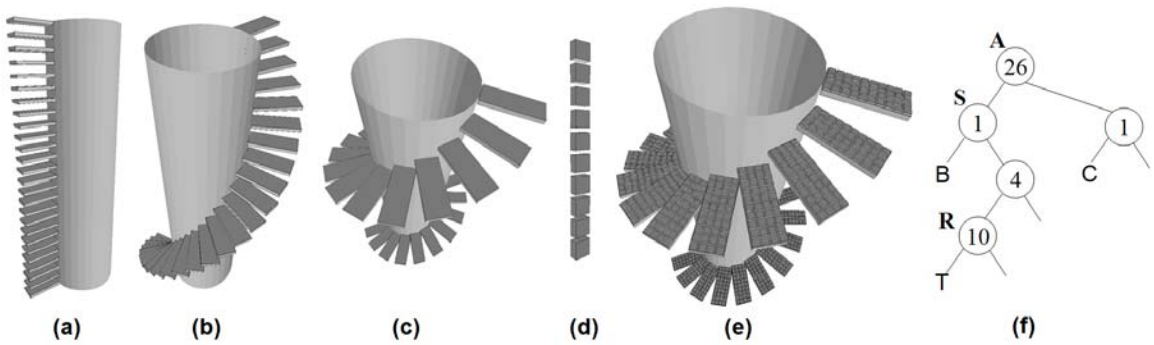

Figure 5.19: Design of a spiral staircase. (a) Vertically ascending steps with no tiles next to a column are defined as A = 26S + C, S = B. (b) Adding small rotation around column forms screw mortion. (c) Further rotation around column is added. (d) A row of tiles is defined as R = 10T. (e) The final model defined as A = 26S + C, S = B + 4R, and R = 10T. (f) The corresponding graph has five pattern nodes and three primitive nodes.

5.5.3 Abstract art

Here we consider a classic recursive example, a variation on the Sierpinski gasket (Figure 5.20). Given a designer specified expression $S = C + 3S^4$, a modeling system, e.g.

[van Emmerik et al. 1993], might show 121 instances of a cylinder. The designer clicks on the "+" symbol to select the first instance (at the top recursion level) of the pattern 3S and adjusts its frame F_2 by rotating it 90 degrees left, uniformly scaling it by one half, and then translating it 1.5 units to the left. The system computes the grouping transform $t_4 = F_2 \cdot F_1^{-1}$. Now the designer selects the pattern "3S" and the system selects the second instance (at the top recursion level). By adjusting the frame F_3 of the second instance, the user can adjust the pattern transform which is calculated as $t_5 = F_3 \cdot F_2^{-1}$. Thus, the designer is able to define and adjust the arrangement of 120 instances with respect to F_1 by adjusting just two other frames F_2 and F_3.

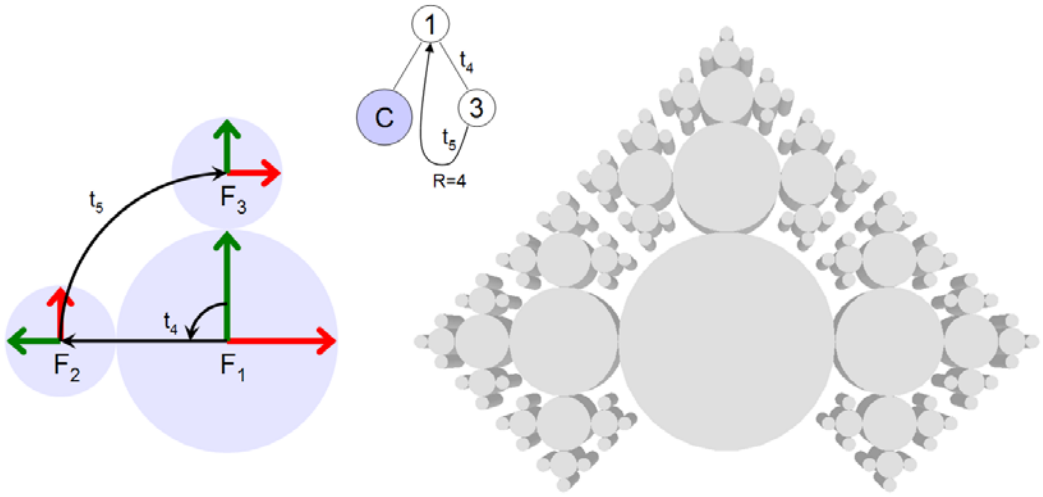

Figure 5.20: A Sierpinski-inspired gasket is defined using the expression S = C + 3S^4. The 121 instances can be arranged by positioning three instances (corresponding to frames F_1, F_2, and F_3).

5.5.4 Grove of trees

Finally we consider an example with patterns of recursive components. The designer designs a grove G = 3R as a pattern of three rows of trees, each of which is a pattern R =

4T of four trees. When a feature is still undefined, placeholders for the frames can be displayed (Figure 5.21 left). Now the designer defines a tree $T = B + 3T^3$ as a branch plus a pattern of three "trees" which recursively defines more branches therein. The resulting model (Figure 5.21 middle) consisting of 480 instances of the branch B is represented by a graph with four pattern nodes, one recursive link, and one primitive node (Figure 5.21 right).

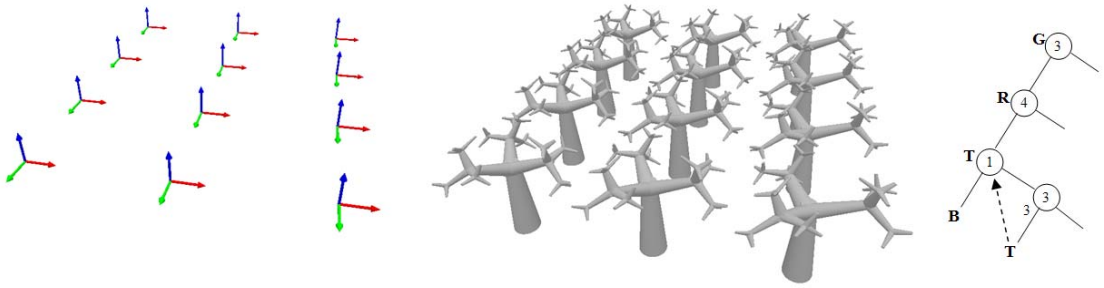

Figure 5.21: When a feature is still undefined, placeholders for the frames are displayed (left). A grove (middle) is defined using the expressions G=3R, R=4T, $T=B+3T^3$. The corresponding graph is given (right).

6 EXCEPTIONS

6.1 Introduction

6.1.1 Overview

Hierarchies of regular patterns can be used to provide a concise representation for models with repetitive or recurring elements. It also provides a means to reduce laborious repetitions from the design process. However, often the placement, shape, or even existence of a selection of the occurrences in the pattern must be adjusted. For example, the presence of a pillar may require that we remove the same chair on each floor of a building (Figure 6.1a). Or one may wish to move the last chair of each dining set so that it faces the other chairs (Figure 6.1b). We call such a deviation from the regular form an *exception*.

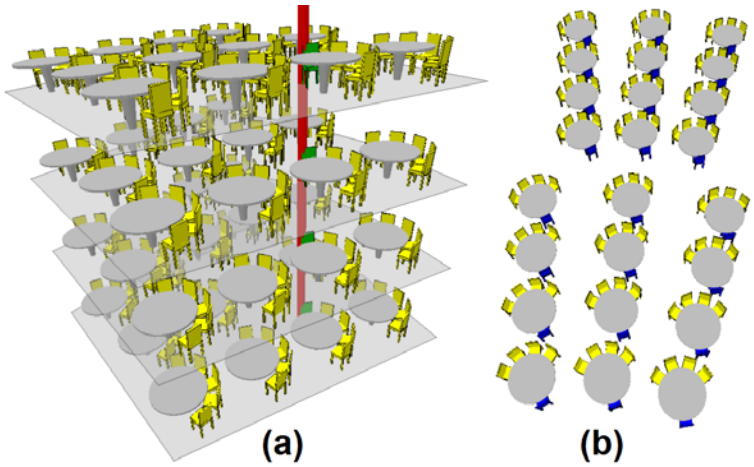

Figure 6.1: Example exceptions. (a) The same chair on each floor (in green) needs to be removed due to a column (in red). (b) The fifth chair (in blue) at each table has been rotated to face the others and tucked under the table.

To specify an exception one must indicate which occurrences are to be adjusted and how to adjust them. Thus, we may define an exception to be a selection plus a treatment. An *exception selection* is the set of components to be modified and an *exception treatment* is the modification information (e.g. displacement, deletion, color change, etc.) to be applied to that set. Once a selection of components is made, any number of treatments can be applied to that selection. Hence, we will focus on giving the user a way to make a selection and will not discuss the possible treatments.

6.1.2 Problem statement

The objective is to support exceptions in pattern hierarchies. More specifically, given an assembly hierarchy as described in this thesis, we develop an approach that facilitates the selection of subsets of instances which reflects the semantics of the hierarchy that describe and organize the components in the assembly. This includes: (1) a representation for maintaining the selection of the exceptions, (2) algorithms for identifying, processing, and rendering an assembly containing specified exceptions, and (3) a user interface to approach to support effective user selection of subsets and editing of selected subsets of instances.

6.1.3 Challenge

For an assembly that includes patterns, making a selection can imply picking many items. Furthermore, the nesting of patterns results in a multiplicity of instances from which to choose. For example, in the building model one can select chairs within a table, down a row of tables, across rows of tables, across floors, or across some combination of

these dimensions. This multiplicity is exacerbated with a recursive pattern assembly. The flexibility to group patterns as assemblies allows for more complex models but also adds further complexities to the selection process. For example, we would like to support selections of the item with the same semantic position in a group such as the table within a dining set group. Besides the challenges in multiple object selection that come from nesting, recursive nesting, and grouping, there are other complexities including the ability to select from different types of components, e.g. chairs or tables, and the ability to make compound selections, e.g. a set of chairs with a table as one selection unit. In summary, we would like to support selections based on all or as many as possible of the semantic relationships represented in the scene or assembly model.

6.1.4 Goals

In order to consider an approach successful, we set forth a number of goals as follows.

Concise representation. The representation should be concise and compatible with the assembly graph. Since the assembly graph contains information on the designer intent, an exceptions representation which closely follows the graph will be easier to maintain while the user edits the model.

Efficient computation. The computation of the selections should be efficient in order to support interactive manipulation and feedback for the user. Efficient computation can also open up other opportunities for user support including context highlighting and selection guides, e.g. mouseover preview of selection. Furthermore, on the developer's side, a simple solution is easier to implement and integrate into more complex scene and assembly representations.

Intuitive user interaction. The approach should support direct and intuitive user interaction. User input should affect the model in a predictable yet adjustable way, both in the changes stored in the representation and in the feedback provided or made available to the user.

Relevant subsets. The subsets which the user can select should be relevant and useful to the user. In other words, we do not need to support selection of arbitrary obscure subsets of instances, but rather should support selection of subsets which reflect the underlying designer intent contained in the assembly graph representation. Hence, this coherence may contribute to predictable and adaptable user interaction while specifying subsets.

6.1.5 Multiple Object Selection (MOS)

The most common approaches for multiple object selection (MOS) include serial selection techniques that require the user to select objects one at a time, e.g. the ubiquitous ctrl+click (or shift+click) approach, and parallel selection techniques such as brushes, lassos, and selection shapes. However as Lucas et al. [2005] point out, each has certain limitations, especially in 3D. For instance, multiple objects may be difficult to distinguish, isolate, or even see due to occlusion, rendering size, environment clutter, and other display factors. Requiring the user to adjust the view can be tedious, cumbersome, and even burdensome, especially when the number of objects to select is high, and may still fail to make certain objects accessible. Systems commonly address this issue with an indirect selection technique, that is, by allowing the user to specify the desired selection using an alternate representation such as a model tree or component list. Some systems allow selection by common attribute or provide a more general selection query or search

[AutoCAD, ProE, Miller and Myers 2002]. Such indirect selection techniques are useful, but are generally abstract and less intuitive than direct manipulation techniques.

Oh et al. [2006] describe an approach for selecting objects in groups. Their approach relies on dynamically computing a group hierarchy based on the notion of gravitational proximity using heuristics such as contact or intersection and factors such as speed and direction of mouse drag. Their approach does not rely on semantic or user specified information for structure and is appropriate for dynamic environments or situations where flexibility is required. It is less appropriate for rigid and exact specification of selections, particularly when objects or components are frequently or always in contact with or intersecting each other, e.g. when modeling parts, assemblies, and structures, or with CSG and feature-based modeling.

6.1.6 Our subset selection approach

We introduce an approach called OCcurrence selecTOR, or OCTOR [Jang and Rossignac 2008], for making selections of multiple components in a scene or assembly. Our approach is applicable to pattern hierarchies, which semantically relate the many components of the model. Hence, it is appropriate to refer to our approach as a subset selection approach since it operates in the context of coherently related patterns rather than more generally or arbitrarily organized collections of objects.

Our approach naturally supports direct selection of groups of occurences, yet requires picking only two of the objects/occurrences to be selected. The general idea is to have the user directly pick two of the occurrences and let the system decipher the rest in a way that is predictable and repeatable. At the same time, the proposed system allows for iterative refinement which can be guided or scaffolded. For example, if the two picked instances

do not yield the desired selection, the user may click additional instances to adjust the selection. If the user then reverts back to two previous picks, the resulting selection is the same as the previous time those same two picks were chosen, regardless of the order they were picked. We show that making two picks is sufficient to establish any selection in a coherent and relevant subset of instances with respect to the pattern hierarchy. In particular, subsets which include all instances which have one or more specific positional attributes in common with respect to the pattern hierarchy. For instance, the second chair at every table on the first floor of the bar scene uses chair position at a table and floor of the bar as the attributes in common while which row of tables on a floor and which table on a row the chair belongs to are attributes without constraints. We call such a selection subset an *OCTOR selection*. The aim is not to replace other selection techniques but to give users another option, which in certain cases is more intuitive, efficient, and accurate.

On the developer's side, the OCTOR representation provides a compact encoding for multiple occurrence selections and is easy to compute. It does not require expanding the graph into a tree or storing an explicit list of the selected occurrences and is simple enough to be extensible and combinable. For instance, multiple OCTOR selections can be combined in a list or with Boolean operations.

Furthermore, we show how the OCTOR approach can be used to support MOS in pattern hierarchies with recursive definitions (Figure 6.2) [Jang and Rossignac 2009]. In particular, we show that the same basic approach for selection encoding and processing applies and only simple modifications to the scene graph design are required to support recursive structures.

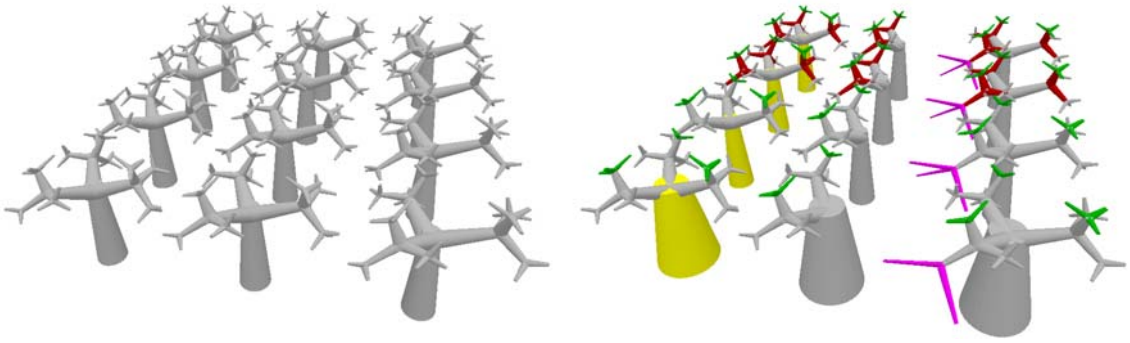

Figure 6.2: A pattern hierarchy with recursive structures is used to define rows of trees (left). A series of edits on a variety of subsets of components in the scene takes a couple of minutes using the OCTOR approach (right).

6.2 Other approaches to subset selection

Before presenting our solution, we discuss the benefits and drawbacks of three simple techniques for specifying and representing exception selections in pattern hierarchies.

6.2.1 Graph expansion

The first possible approach is to expand the graph into an n-ary tree (Figure 6.3). The child-nodes of nodes with pattern-count larger than one are replicated, replacing the n.L link with n.o such links. This expansion is performed recursively and includes recursive chains up to the recursion limit. In our example of a bar, such an expansion would produce a tree with 24 table leaves and 120 chair leaves. In the simple recursive pattern defined by M=C+pMr, an expansion would produce a graph with p^{r+1}-1 leaves of C, an exponential growth. The designer would then be able to select individual leaves one by one and adjust their poses or attributes. This approach has the drawback of increasing storage and of not preserving the structure of the pattern hierarchy, which represents the designer's intent [Rossignac et al. 1988] and hence should be preserved to facilitate

further editing. For example, the designer may later decide to add a third floor or to squeeze in more chairs at each table. Even with an approach based on partial graph expansions, managing change can be challenging and maintaining certain selections may still require an external structure [Rappoport 1993]. Hence the remainder of the paper is focused on approaches that do not require such a graph expansion.

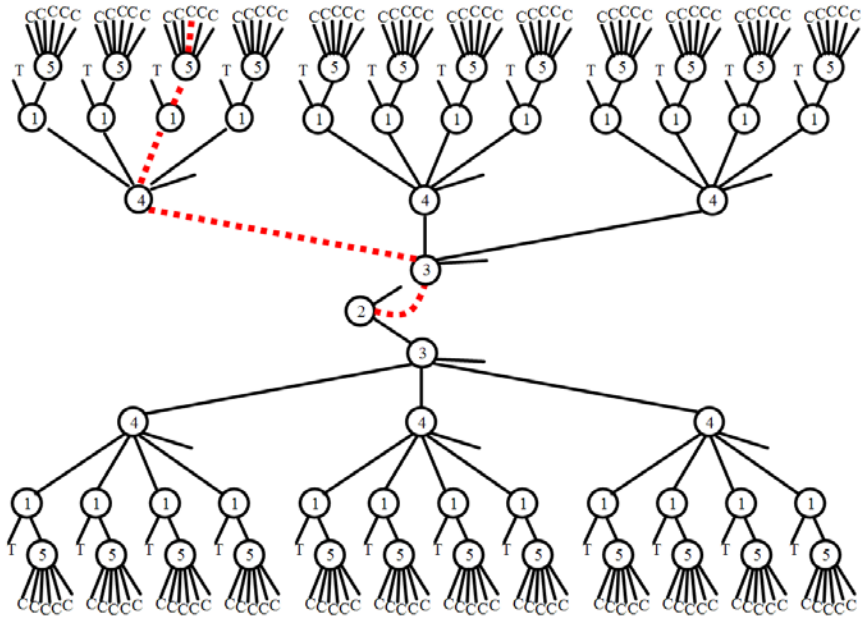

Figure 6.3: Expanded graph of the bar scene with path "21304" highlighted.

6.2.2 Path naming

Note that each leaf in the expanded graph is an occurrence of a component. Each leaf may be represented by a *path*. The path is the concatenation of integers, each specifying which link is followed from a node to its child. The order of these integers corresponds to the traversal of the expanded graph (including recursive expansions) from the root to the desired leaf. When the path follows link K from a node n, we append `0' to the path when

K is the link from n to n.R (corresponding to the grouping link) and `k' (where k=1...n.o) to the path when K is the link from n to n.L (corresponding to the patterning link). For instance, the path "21304" corresponds to following patterning links 2, 1, and 3, the grouping link 0, and then patterning link 4 in the expanded graph of the bar scene (Figure 6.3). Note that such a path uniquely identifies a component, even in a recursive hierarchy. For example, the path "03020202" corresponds to the right most disc in the Sierpinski gasket in Figure 5.20.

The notion of a path suggests an alternative approach where one represents each occurrence by its path in the non-expanded graph. For example, the highlighted chair in Figure 6.4a corresponds to path "21304".

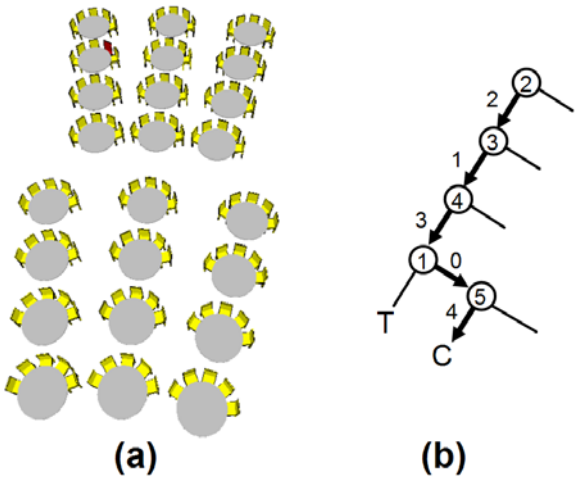

Figure 6.4: (a) The highlighted chair is identified with path "21304". (b) The path is illustrated on the unexpanded tree.

In this approach, the designer would manually select each chair that should be treated as an exception and would specify the associated exceptional treatment. A list of exceptions (selections plus treatments) is stored in a separate structure from the graph.

This approach avoids graph expansion while still allowing arbitrary selections; however, it still has the drawback of requiring a manual selection and explicit storage of each occurrence in the exception selection set. Hence, to further reduce the designer's labor, we will develop an implicit approach where the designer will not, in general, need to select each exception instance. While storage is not a big issue, our approach also has the benefit of compact storage.

6.2.3 Partial path naming

A third approach would be to ask the designer to associate each exception with a node n in the graph and to represent the set of target occurrences by a partial path in n. For example, every occurrence of the fourth chair of each set would be identified by (S,"04") (Figure 6.5ab) and the every occurrence of the fourth chair in the third sets of each row would be specified by (R,"304") (Figure 6.5cd).

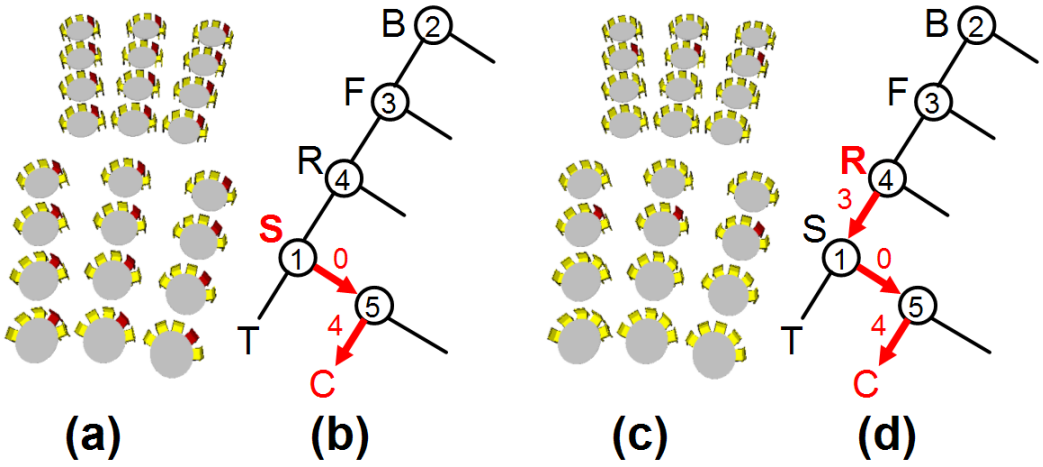

Figure 6.5: The set of highlighted chairs in (a) and (c) are selected with the expressions (S,"04") and (R,"304") respectively. The corresponding partial paths are shown on the graph in (b) and (d).

Even though this approach was successfully used by Rossignac [1986] to specify a set of constraint-satisfying adjustments to features in CSG models, its limitations may require unnecessary replication of the designer's effort. For example, this approach would not allow us to select the fourth chair of the third set on each row of the first floor (Figure 6.6a), because (R,"304") does not let us differentiate floors (Figure 6.5c) and because (B,"11304"), (B,"12304"), and (B,"13304") would only specify a single chair each. To obtain the desired selection would require the union of the three individual selections (Figure 6.6bcd).

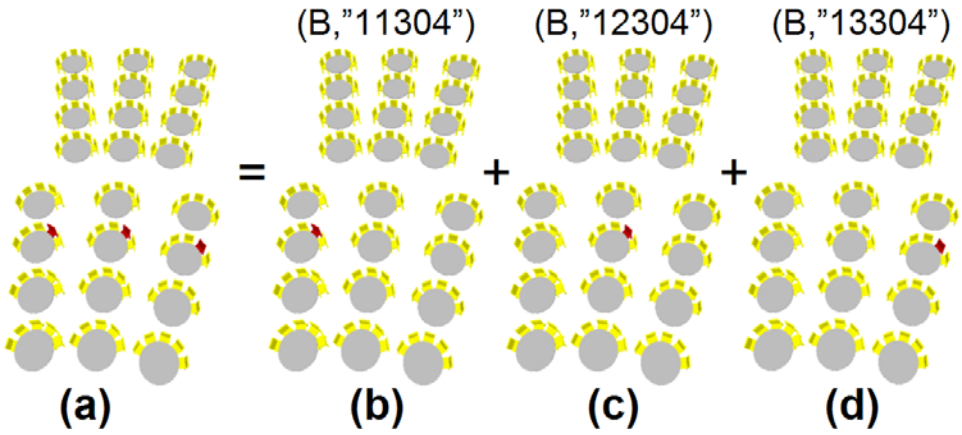

Figure 6.6: The set of highlighted chairs (a) cannot be specified using a single partial path but requires the union of three partial path selections (b), (c), and (d).

6.3 OCTOR selections

We now describe our solution, which does not suffer from the limitations of the three approaches discussed above, and which offers several advantages: conciseness of representation, elegance of the user interface and reduction of the required user actions and cognitive burden, and increased generality. We begin by describing a concise

representation for selections based on wildcards and show that it supports an elementary and essential set of selections. Later we show how the approach simplifies the user interaction required to make a selection.

6.3.1 Wildcards

As discussed earlier, we do not want to represent the group of selected occurrences by a list of paths and may not be able to represent important sub-patterns by a single node name and partial path. Instead we propose to represent a selection by a path to any one of the selected occurrences (the one clicked by the designer) and by a mask string of bits, one for each link on the path. A `0' in the mask corresponding to link (n, n.L) indicates that the subsequent selection should be applied to all occurrences of n.L. A `1' in the mask indicates that it should be restricted to the occurrence of n.L specified by the path. For example, a `0' bit would let us interpret the second field in the paths (B,"11304"), (B,"12304"), and (B,"13304") as a wildcard and let us interpret this path as "1*304" (using path "1i304" with mask "10111"), hence producing the selection in Figure 6.6a. We refer to such a path string with wildcards as a *wildcard path string* or an *OCTOR path string* and the set of components such a path string selects as an *OCTOR selection*.

Note that only mask fields that correspond to left links (patterning links) are allowed to contain a wildcard `0'. Mask fields corresponding to right links (grouping links) should always be a `1' (signifying a constraint) to respect the unique identity of all the occurrences. This is illustrated in the spiral staircase in Figure 6.7, which is specified as A=26S+C, S=B+4R, and R=10T. Any OCTOR path string referring to the tiles (T) would contain a `1' in the second field of the mask since the expression S=B+4R causes the pattern of 4 rows (4R) to be linked to the right of the pattern of one stair block (B). For

example, the spiral staircase in Figure 6.7c was designed by applying exceptions to selections using the masks “0101”, “0110”, and “1100”. If the expression for the stair (S) was changed to S=4R+B, then the path to T would contain only left links and these three masks would be “001”, “010”, and “100” respectively.

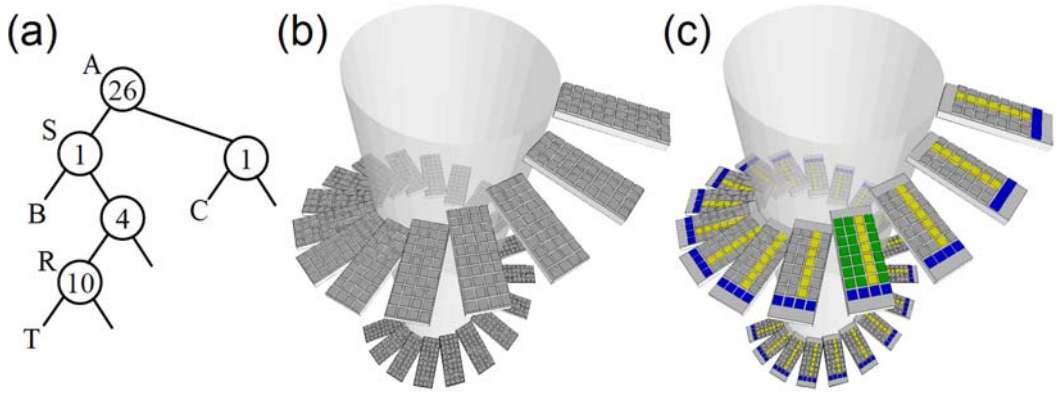

Figure 6.7: A spiral staircase (b) has a pattern of 40 tiles arranged as 4 rows of 10 tiles on each of the 26 stairs and is specified by A=26S+C, S=B+4R, and R=10T. The corresponding graph is given in (a). Selections based on three out of the eight possible selection masks for tiles (T) were used to design the spiral staircase in (c).

To support recursion, consider the expanded graph even though we do not expand it literally. Then, components which were once represented by the same node in the graph are now explicitly different components represented by different nodes and we can use the same path and wildcard technique as the one which we explained for non-recursive patterns. Figure 6.8 shows several examples of OCTOR selections on a recursive pattern hierarchy.

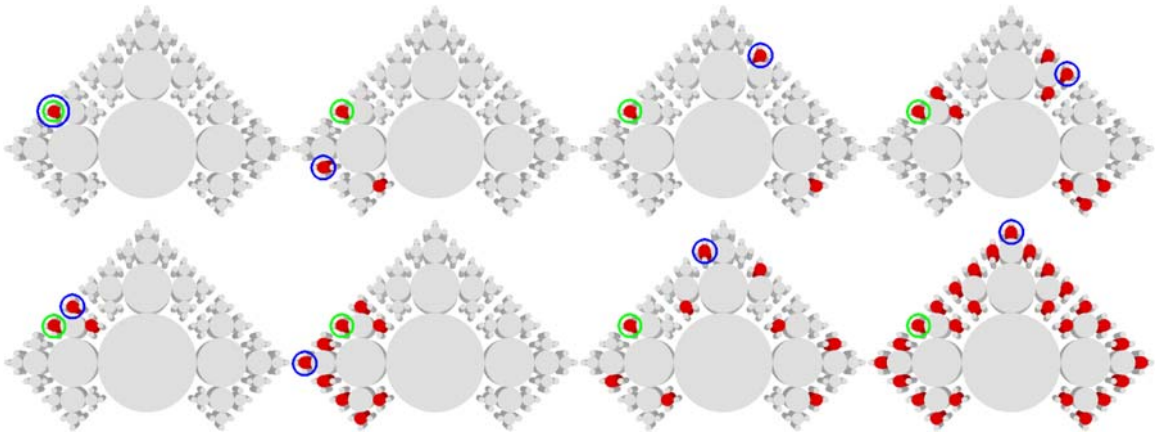

Figure 6.8: Example selections of the depth=3 cylinders from a Sierpinski layout defined by S=C+3S4. The paths are "0103011", "010*011", "0*03011", and "0*030*1" on the top row and "01030*1", "010*0*1", "0*0*011", and "0*0*0*1" on the bottom row. Examples of possible user clicks to obtain each selection are indicated by circles.

6.3.2 Exception culling

Once the user defines a selection set, the user specifies an exception treatment to apply to that set. This defines an exception. Now we need to support rendering of pattern hierarchies with exceptions or simply access each occurrence in its final pose (e.g. for picking) or state (e.g. color, deleted, etc.). We may modify eval(), which was presented in Section 5.2, to incorporate the treatment of exceptions. We use a depth-first traversal where we follow the path with respect to the exception selection. If we walk off the selection path, we mark that exception as inactive, visit the rest of the path recursively, and then mark it back as active. When we reach a primitive (leaf), if the exception is active then its treatment is applicable to that occurrence. For tracking a single selection, there is no need to continue traversing a path for which the selection is already marked inactive; however, the idea of *exception culling* can be applied to a whole list of exceptions, not just one as is listed in eval2(). In other words, instead of tracking one path, we simultaneously track all of the paths in a list of exceptions. For each path in the list of

exceptions, we check to see if we walk off that path and mark it as inactive if we do. We continue traversal down a path if there are any active paths remaining in the list of exceptions. At any point in the traversal, all currently active paths in the list of exception paths indicates that the corresponding exception is applicable to the occurrence represented by that node. Hence, exception culling also supports exceptions that are not applied at leaf nodes. All active exceptions are potentially valid at any given node during traversal and a simple node id check is needed to confirm that it is applicable to the node.

```
eval2(n, r, selected) {
  boolean deactivated = false;
  if (isPrimitive(n)) process(n, selected);
  else {
    pushMatrix();
    for (int i=1; i<=n.o; i++) {
      cullX(i, &selected, &deactivated);
      eval2(n.L, r+1, selected);
      restoreX(&selected, deactivated);
      applyMatrix(n.l);
    }
    popMatrix();
    pushMatrix();
    applyMatrix(n.r);
    cullX(0, &selected, &deactivated);
    eval2(n.R, r+1, selected);
    restoreX(&selected, deactivated);
    popMatrix();
  }
}
cullX(i, *selected, *deactivated) {
  if (*selected && (mask[r] == 1) &&
     (path[r] != i)) *deactivated = true;
  if (*deactivated) selected = false;
}
restoreX(*selected, deactivated) {
  if (deactivated) selected = true;
}
```

Since the mask only requires one bit per link, we can incorporate it into the path string by using the sign bit. Practically, we can adopt an even simpler encoding such that path[j]=-1 when mask==0, thus the path value is -1 for wildcards, 0 for going right, and 1...n.o for going left. We trivially modify cullX() to incorporate this simplification.

```
cullX(i, *selected, *deactivated) {
  if (*selected && (path[r] >= 0) &&
     (path[r] != i)) *deactivated = true;
  if (deactivated) selected = false;
}
```

6.3.3 Equivalence

One may think of the OCTOR selections as representing all the axially-aligned slices of 0...p dimensions through discrete p-dimensional space, where p is the number of patterning links (i.e. left links) and thus represents the pattern nesting depth. Because each possible bit mask represents an equivalence class of slices (Figure 6.9), it may be convenient to keep the bit mask and the path string separate for certain applications. For example, the user can make a complex selection on components that are obscured or difficult to visualize by first specifying the selection pattern (slice) on more accessible occurrences, on an alternate representation, or on another model. Then, the user can apply the slice somewhere else using one click (a single pick). In other words, the user first specifies and fixes the mask as constant and then only needs to choose a single additional occurrence to specify the path. The user can use this technique to, for example, explore the selection space or underlying organization of the instances.

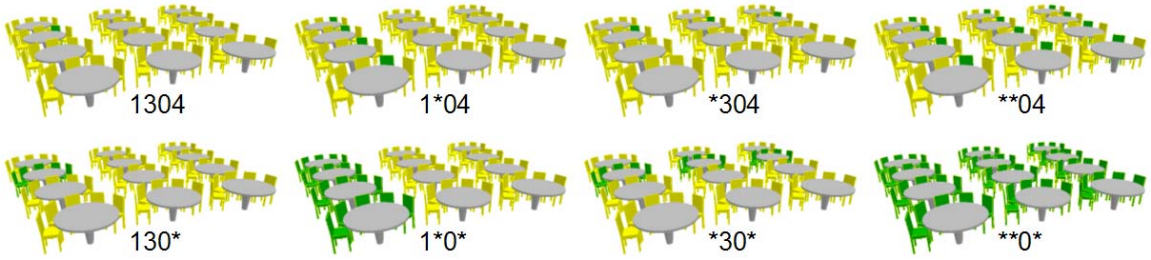

Figure 6.9: The 8 equivalence classes of selections for one floor of the bar are shown. The highlighted chairs are selected using path "1304" with 8 masks "bb1b" where b is a binary digit `1' or `0' starting with "1111" (first row left) and ending with "0010" (bottom row right).

Only a small subset of the 2^r (where r is the total number of occurrences of a given component, e.g. a chair, a table, a dining set, etc.) possible selections can be represented as a single OCTOR selection. For example, the selection shown in Figure 6.10 cannot be specified by a single OCTOR path string but would require, for example, the union of three or the subtraction of one from another. Furthermore, "patterned" selections such as the black tiles of a checkerboard or a line of diagonal tiles will not benefit from OCTOR.

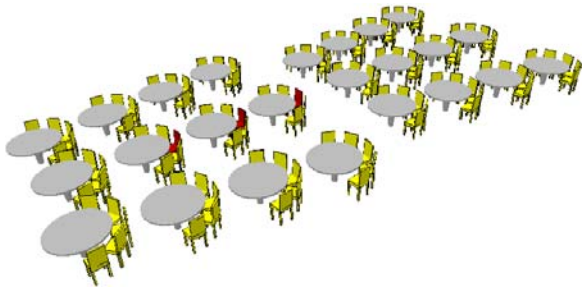

Figure 6.10: The selection cannot be specified by a single OCTOR path string. Using Boolean operations, it can be specified in 2 strings with the expression "12*03" minus "12103" as opposed to 3 strings using a list of paths.

Nevertheless, we feel that the selections directly accessible through OCTOR provide a valuable addition to other mechanisms discussed above. An OCTOR path string allows us to make generalizations (using wildcards) and constraints (path position with no wildcard) directly corresponding to the pattern hierarchy which represents the designer intent. Of course several OCTOR selections may be combined (union) to produce more elaborate sets. Furthermore, one may envision a more general scheme offering Boolean operations on selected sets, where the selections directly accessible through OCTOR provide an elementary set of selections which can be combined to form others.

6.4 User specification of OCTOR selections

Our basic approach for specifying a selection is to obtain the path from the first click (the first pick) and to obtain the mask from subsequent interaction such as additional clicks (or picks). To support a graphical user interface for OCTOR, we need to: 1) allow the user to pick individual occurrences, 2) compute the path of a picked occurrence, 3) produce a candidate set for further refinement, and 4) use these three tools to let the user interactively build a selection mask.

6.4.1 Picking and computing a path

To compute a path, we simply use eval(n) while tracking the path as follows:

```
eval(n, path, r) {
  if (isPrimitive(n)) process(n);
  else {
    pushMatrix();
    for (int i=1; i<=n.o; i++) {
      path[r]=i;
      eval(n.L, path, r+1);
      applyMatrix(n.l);
    }
    popMatrix();
    pushMatrix();
    path[r]=0;
    applyMatrix(n.r);
    eval(n.R, path, r+1);
    popMatrix();
  }
}
```

In process(), we need to decide if the current occurrence corresponds to the user click position. There are various solutions to this basic picking problem [Lucas et al. 2005] including ray-casting, which works even for occluded objects and z-buffer selection, which may be combined with stencil planes and peeling [Hable and Rossignac 2007] to select hidden instances. If the user wants to select an occluded object, additional steps are required, making it impractical for MOS, where many picks may be required.

In the situation where picking and disambiguating individual components is cumbersome, picking multiple instances would be increasingly burdensome. Our MOS approach has the advantage that it only requires a single pick to be disambiguated since subsequent picks are on occurrences of the same pattern component. After resolving a single pick, these related occurrences can be visually isolated, for instance, by hiding all other objects. If none of the desired occurrences are visible, the designer may select the primitive in the text representation (or a component list, tree, or graph) to temporarily hide all others. For example, a subtracted CSG component located inside of another may require indirect selection of the first click. After that, the other instances are made visible and other components made invisible or diminished. Subsequent clicking can occur directly on the scene. Thus, our approach supports an interactive and exploratory approach to making a selection.

6.4.2 Building a selection mask

When the user makes the first pick, this defines a path of length d. The user now needs a way to specify a wildcard mask of length d.

A naive approach is to specify the d bits using d clicks. Each time we ask 1 question by proposing a candidate set. The user would choose Y or N to decide if they wish to toggle that bit resulting in the selection of the candidate set.

A more direct approach is to allow the user to directly click on additional occurrences, i.e. a second, third, etc., and have the system guess or infer the selection from the **cumulative** set of picks. For example, when the designer selects chair “11304” and “12304”, the system generates the mask “10111” producing the selection in Figure 6.6a.

Adding a third chair “21304” results in a mask of “00111” (Figure 6.5c) and adding a fourth chair “11204” results in a mask of “00011” (Figure 6.5a).

For our approach, observe that among the set of paths, path fields that differ indicate generalizations and fields which are identical indicate constraints. Thus we see that only the latest pick along with the first is necessary for specifying a path plus mask. In fact, any selection representable by a single OCTOR path string can be specified with only two clicks. For example, to specify “**304” (Figure 6.5c) the user can select chairs “12304” and “23304”, which results in path “12304” with mask “00111”. Thus we propose to keep track of only the **first and latest** pick.

Our basic selection approach can be summarized as follows: (1) The first click picks one instance as path P_1. By default we may initially assign the second pick as $P_2 = P_1$, thus selecting one instance. (2) The second click picks P_2 and enlarges this selection as path P_1 (or P_2) with wildcards where $P_1 \neq P_2$. (3) A third click (and subsequent clicks) **replaces** P_2 and recomputes the selection. Note that this differs from the cumulative approach where the third click would pick P_3 and then compute the selection using all three picks. Thus our approach has the advantage that there is less modal state for the user to track and manage. Another advantage of our approach is that it is intuitive. The user directly clicks the occurrences in the desired selection set and the system updates the highlighted set. Thus the user may interactively refine their selection by selecting alternative occurrences.

Notice that if we expand the recursive links of the graph, the result is simply a hierarchical pattern. Thus, the same basic selection approach applies to recursive patterns.

Figure 6.8 gives eight examples of selections which can be made with two clicks on a recursive pattern.

6.4.3 Refinement set

A direct clicking approach requires the user to find the correct occurrences to click. While the user can interactively sample the selection space by trial and error, the system may be able to help the user by identifying a small set of occurrences to click. For example, after the user clicks one chair in the bar scene there are only 16 OCTOR selections possible which use the path of the picked chair. (These correspond to the 16 possible masks which correspond to the 16 different equivalence classes of selections.) Yet the user can click any of the 120 chairs to make one of these 16 selections. The system can help lessen the user burden of identifying occurrences to click by presenting just one option for each of the 16 equivalence classes (Figure 6.11a). Fortunately, this is straightforward to compute [Jang and Rossignac 2007]. For each possible bit mask, construct a path string which is the same as the path of the first pick in all fields except wildcard fields. Any variation of the field value within the pattern-count range corresponding to that field is acceptable. For instance, on may use the next or previous value as a simple heuristic. This has the benefit of near access to far selections. That is, the user can select occurrences located far apart by picking occurrences close together (i.e. close to the first pick). For example, on an airplane seating example, the user can make any OCTOR selection (i.e. single, row, column, all) by clicking on 2 of 4 seats (Figure 6.12). Furthermore, it simplifies making selections when certain occurrences are obscure or hidden, e.g. when the placement of components in the assembly or scene are dense,

occluding, or arranged in configurations that are difficult to visually disambiguate (Figure 6.13).

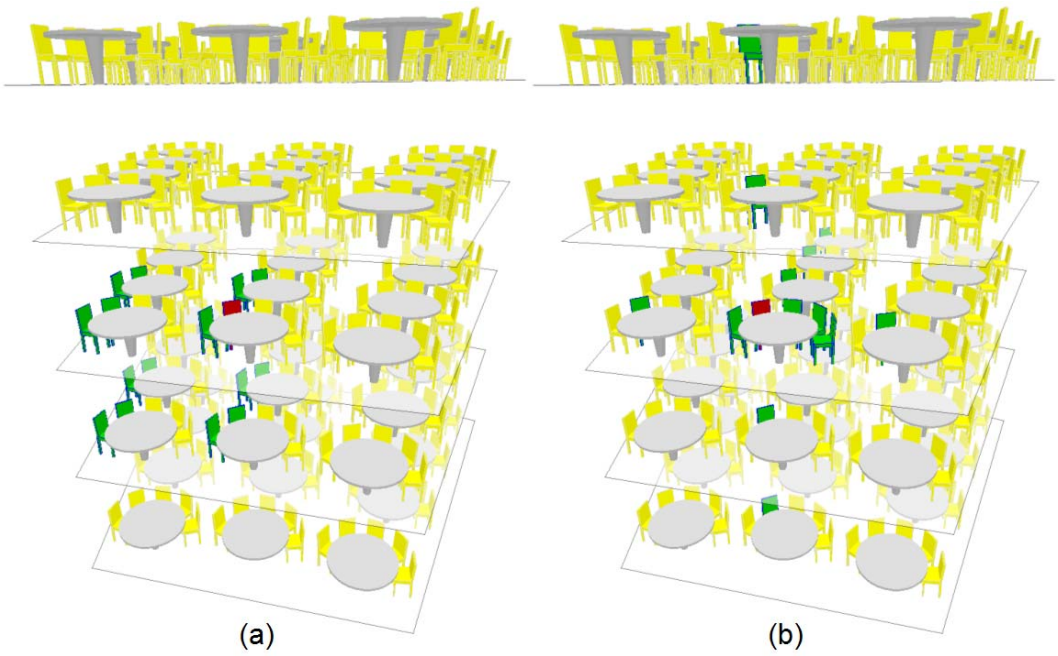

Figure 6.11: Refinement guides for a 5-floor version of the bar scene given first click selection "32102": (a) 16 unique selection classes. (b) coherence in one path field.

An alternative approach to guide the user is to highlight one or all occurrences with paths that vary in only one field (Figure 6.11b). This visual guide helps the user choose and maintain generalizations or constraints when endeavoring to expand or refine the selection. For example, to specify selection "***04" (Figure 6.5a), the user first selects chair "12304" and then needs to find a chair that is on a different floor, row, and set. This guide makes it clear which chairs are on the same floor, row, and set and thus helps the user find one that differs.

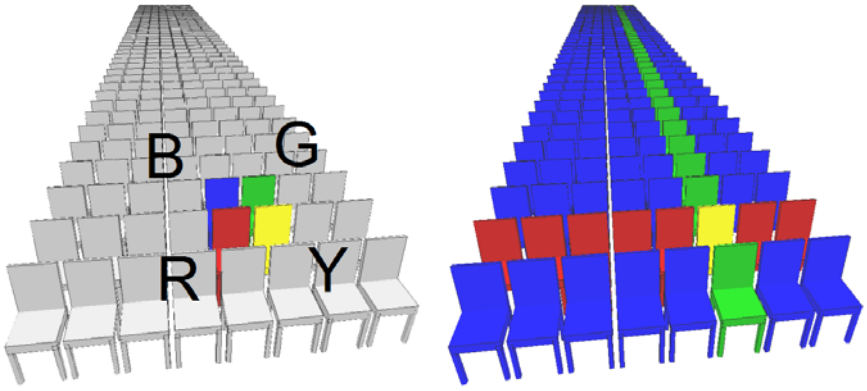

Figure 6.12: Clicks on just 2 of only 4 seats (left) are required to select a single seat (Y, Y), a row (Y, R), a column (Y, G), or all seats (Y, B). In fact, after clicking seat Y first, a second click on any other seat in the same row selects the row, any other seat in the column selects the column, and the rest of the seats select all (right). This selection principle extends to n-dimensions.

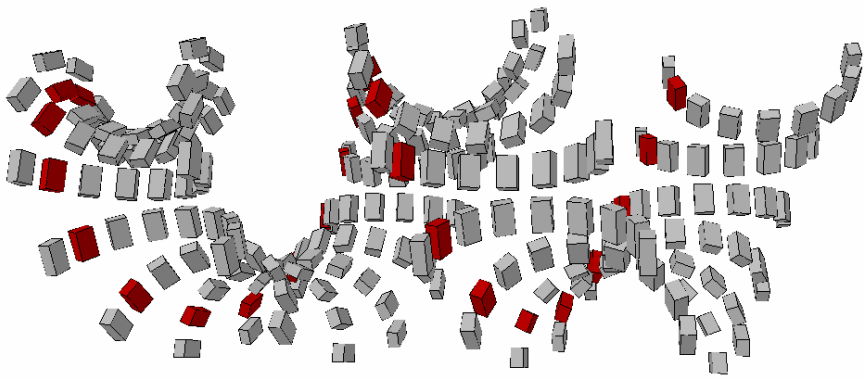

Figure 6.13: Specifying a selection (highlighted) on this helix (defined as a spiral of a semicircle of blocks) would be complex without the two-click approach and can benefit from selection guides.

6.4.4 Picking compound components

The OCTOR approach can be used to make selections of multiple compound components, not just of simple components. For example, in the bar scene, an entire dining set can be selected initially as the basic selection unit by the user clicking on a table and a chair at that table. The path for this initial pick is simply the path to the lowest (deepest) common ancestor of the multiple components identified. Then, subsequent clicks on either tables or chairs can be used to pick more dining sets, i.e. to define the

selection mask. The wildcard path for a subsequent pick is computed by truncating the path to match the length of the initial pick and applying the same algorithm for computing simple component selection masks.

6.4.5 Picking across multiple recursive levels

Recall that a wildcard generalizes the virtual link that would result from traversing and expanding all the recursive links. A single OCTOR path string can only represent a subset out of the set of all occurrences derived from a single node in this virtually expanded graph. Occurrences derived from different nodes (which are, more specifically, on different recursive levels) are considered to be different entities. For example, a single string cannot represent, say, all of the left ears from the largest down to the smallest in the mouse ears example (Figure 6.14). However, we propose to remove this limitation with the following extension. We consider three cases given two path strings corresponding to two picks of the same component, i.e. corresponding to the same node in the unexpanded graph: (1) If there is no recursive link on either path then the standard method already described is used to compute an OCTOR path string from the two path strings. (2) For two picks of the same component on the same level of a recursive pattern (i.e. both path strings have recursive links and refer to the same virtual node of a virtually expanded graph where all recursive links are expanded), the computation also uses the standard method. (3) For two picks of the same component but on different recursive levels (say, level m and level n with n>m) of a recursive pattern, we give the user four options.

(A) Go up to the same level (the shallower one, m) and use the standard approach. This is achieved by truncating the longer path to the length of the shorter one and then computing the wildcard path string using the standard method.

(B) Alternatively, we can replicate the extra part (i.e. links with traversal depth > m) of the path of the longer one and append this to the shorter path string. The wildcard path string is then computed using the standard method. Both options (A) and (B) allow for "lazy" selection of occurrences on the same recursive level but do not support selections spanning multiple recursive levels.

(C) Another approach is to select the entire subtree of the lowest common ancestor. The path to this ancestor is the leading part of the two paths that is identical. Due to exception culling, this will result in keeping the exception alive for all nodes in this subtree including recursive expansions. A simple check of the node id ensures that the exception is applied only to instances of the desired component. The result is that every component that is represented by the same node in the unexpanded graph is selected if it has a recursion depth >= m.

(D) As a refinement of approach (C), we would like to use the deeper path to constrain the traversal with respect to the recursive branching but to still visit all other nodes on non-recursive links. We achieve this by only checking whether we are compliant with the wildcard path string for recursive links. Meanwhile, we allow the traversal to go down links off the path on non-recursive links without deactivating the exception and confirm the correct component using the node id. This approach allows us to make coherent selections of "veins" down the recursive "branches", including the selection in Figure 6.14b.

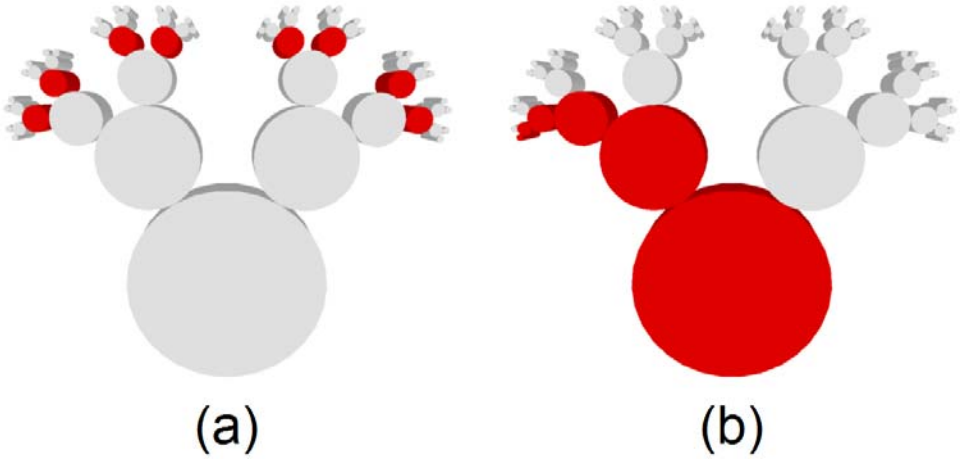

Figure 6.14: The selection in (a) can be defined with a single OCTOR string ("0*0*0*1") while the selection in (b) cannot ("1", "011", "01011", "0101011", "010101011", "01010101011"). The model is defined by M=C+2M5 and has the same graph as the one in Figure 5.8 but with different parameters.

6.4.6 Consistent interaction

Two-click consistency. Here we show that OCTOR selections are repeatable; that is, given a pattern assembly hierarchy as described in this thesis and two picks (which are defined by two path strings, respectively), the selection is unambiguously defined and hence independent of the order in which the two picks are specified. More specifically, we state this as the two-click consistency theorem:

Theorem 1. *Given two path strings, there exists one and only one resulting OCTOR path string that represents them.*

Proof. This follows by construction. Assume the two path strings A and B have length m and n respectively. To compute OCTOR path string W of length l where l = min(m, n), each corresponding field A_i and B_i are compared and deterministically resolved to be W_i:=A_i if A_i=B_i and Wi:="*" otherwise ($A_i \neq B_i$). Note that this does not mean there does not exist alternate path strings C and D which would generate the same OCTOR path

string. For example, the path string pairs (A="1103401", B="1202401") and (A="1303401", B="1501401") both result in the same OCTOR path string W="1*0*401".

Note that the same consistency holds for k path strings. We state this as the following corollary:

Corollary 1. *Given n path strings, there exists one and only one resulting OCTOR path string that represents them.*

Proof. This also follows by construction. Assume the n path strings A_1, A_2, …, A_n have respective lengths l_1, l_2, …, l_n. We compute OCTOR path string W of length $l=\min_{1\le i\le n}(l_i)$ by comparing each field i in every pair of path strings A_j and A_k and deterministically evaluate W_i as follows: (1) $W_i:=A_{1i}$ if $A_{ji}=A_{ki}$ for all $1\le j,k\le n$. (2) $W_i:=$"*" if $\exists j,k$ such that $A_{ji}\neq A_{ki}$ for $1\le j,k\le n$.

Two-click sufficiency. Now we would like to say more about the sufficiency of two clicks. In particular, any OCTOR selection which we can make with n picks where n>2, we can also make with two picks. We state this as the two-click sufficiency theorem:

Theorem 2. *Given a set of n (n>2) path strings {A_1, A_2, …, A_n} which defines an OCTOR string W, there exists a set of two path strings {B, C} which also defines W.*

Proof. We show this by providing an algorithm which computes {B, C} given {A_1, A_2, …, A_n} and show that V=W, where V is the OCTOR string computed from {B, C}. In particular, we assign $B=A_1$ and compute C as follows: (1) $C_i:=A_{ki}$ where $A_{ki}\neq A_{1i}$ if W_i="*". (2) $C_i:=A_{1i}$ if $W_i\neq$"*". In the first case, we can always find A_{ki} such that $A_{ki}\neq A_{1i}$ since W_i="*" implies that $\exists j,k$ such that $A_{ji}\neq A_{ki}$ for $1\le j,k\le n$ and we can set j=1 without loss of generality. In the second case, we can always assign $C_i:=A_{1i}$ when $W_i\neq$"*" since

$W_i \neq$"*" implies that $A_{ji}=A_{ki}$ for all $1 \leq j,k \leq n$. Now since $C_i=A_{1i}=B_i$, it follows that V_i ="*"$=W_i$.

Transparency of representation in hierarchy. Now assume that the user sees the bar scene but is not aware of the scene graph. This would be the case if someone else designed the scene or if the scene was reverse engineered [Thompson et al. 1999; Langbein et al. 2001]. Note that it is possible to define different scene graphs which describe the same scene. If the extracted hierarchy is simply a re-sequencing of the same pattern dimensions, the paths will change but the effect will be transparent to the user (Figure 6.15).

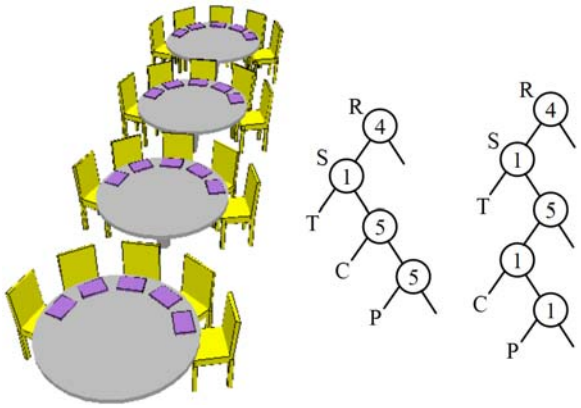

Figure 6.15: A row of dining sets with place settings is defined in two different ways (R=4S, S=T+5C+5P and R=4S, S=T+5N, N=C+P) and yet the differences are transparent to the user when specifying selections of only chairs, only place settings, or only tables. However, only in the second graph can one make a compound selection of a single chair plus single place setting component. In the first graph, one can only select a five chair plus five place setting component as the minimum component containing both a chair and a place setting.

For example, assume a component C can be selected with respect to graph G using string A of length m. Also assume that the equivalent component D can be selected with respect to equivalent graph H using string B of length n. We say that a field A_i ($1 \leq i \leq m$) is a pattern field if it corresponds to a pattern node in graph G. Likewise each pattern field

B_j ($1 \leq j \leq n$) corresponds to a pattern node in graph H. Now if for each pattern field in A there exists a pattern field in B which corresponds to an identical pattern node (same pattern count and transform), we say that the pattern dimensions are re-sequenced with respect to picking C and D. In such a situation, the user can select an equivalent subset of instances defined by OCTOR strings V and W, respectively, by picking the same respective components the scene corresponding to G and H, respectively. The reason for this is that the algorithm for computing the OCTOR path string processes fields independently and the computation does not depend on the ordering of the fields. Hence, the fields which are generalized to be wildcards in A are generalized to be wildcards in their respective fields in B, and likewise for constraint fields. In this sense, we may say that OCTOR selection is resilient to the permutation of the order in which a nested pattern is defined.

6.5 Persistence

In a typical modeling setting, after specifying one or more exceptions, the user may continue to make modifications to the model. For example, the entire bar scene assembly could be repeated (e.g. mall M=7B), more items could be placed on the table as part of the standard dining set (e.g. a lamp with S=T+5C+L), or a design pattern could be placed on the standard chair (e.g. H=C+10P, S=T+5H). Even the pattern counts may be adjusted (e.g. S=T+8C) or components removed completely (e.g. S=5C results from removing the table from the dining set). The challenge is deciding how to update OCTOR paths when the model is edited such that they continue to identify or *name* the same components. Note that the persistent naming problem for parametric, feature, and history based modeling has been well studied [Marcheix and Pierra 2002]. Fortunately, our version of

the problem is less complicated since we directly name the occurrences and their existence or location in the hierarchy is explicit, whereas the structure of topological features may be implicit depending on where and how they are combined.

As described in Section 6.3.2, we store a collection of exceptions as a list of path strings (defining the selection subset) along with a corresponding exception treatment (defining the modification to the selection subset). Hence, we need to maintain exceptions as a separate task as the graph changes. While the fact that exceptions are not part of the generation tree can be considered a weakness of our approach, the exceptions are closely integrated with the tree in our evaluation approach. Furthermore, the changes that would require processing to update the path strings are related to more fundamental changes to a parametric design. The modification of parameters such as pattern count and spacing, e.g. to tweak parametric designs, are more frequently employed by the designer. As we indicate here, it is not necessary to perform additional processing to maintain exceptions under such edits.

6.5.1 How to maintain exceptions

Now we list the basic modification scenarios based on our pattern assembly graph representation (Section 5.4.3) and describe how an OCTOR path string is modified to maintain its selection after the modification. For each we give an example. (The target component for each example path is underlined.)

(1) Change pattern count, e.g. S=T+5C → S=T+8C: no change required.

(2) Change recursion depth, e.g. M=C+3M5 → M=C+3M3: no change required.

(3a) Delete branch not in path, e.g. S=T+5C+L → S=T+5C: no change required.

(3b) Insert branch not in path, e.g. S=T+5C → S=T+5C+L: no change required.

(4a) Delete right branch in the path, e.g. S=T+5C+4L → S=T+4L: delete field in all affected path strings, e.g. …004 → …04.

(4b) Insert right branch in the path, e.g. S=T+4L → S=T+5C+4L, insert `0' (go right) in path position of inserted node, e.g. …04 → …004.

(5a) Delete left branch in path, e.g. R=4S, S=T+5C → R=4S, S=5C: delete field in all affected path strings, e.g. …405 → …45.

(5b) Insert left branch in path, e.g. R=4S, S=5C → R=4S, S=T+5C: insert `0' (go right) in path position of inserted node, e.g. …45 → …405.

(5c) Insert node with left child in path, e.g. B=5F → M=7B, B=5F: insert `-1' (wildcard '*') in path position of inserted node, e.g. 5 → *5.

(6) Delete component, e.g. S=T+5C → S=T: A deleted leaf means all associated exceptions can be deleted.

In case (1), even if we change the occurrence count from 5 to 3 and yet have an OCTOR path string which selects the 4th occurrence, we can leave the OCTOR path string unchanged. The exception culling will deactivate this exception when visiting the other branches (1st, 2nd, and 3rd) and simply will not reach the branch (4th) due to the new occurrence count 3. If the user subsequently increases the occurrence count to include the branch, the exception is maintained. Similarly in case (2), if we expand the recursion and consider this expanded graph, a change in the recursion count either inserts or removes a repeat section of nodes at the tail end of this chain. So as in case (1), the exception handling will ignore exceptions for presently non-existent virtual nodes because they will not be visited.

6.5.2 Implementation

To support persistence given the basic modifications, we implicitly identify the different scenarios by running a path-following traversal with modified exception culling. Instead of maintaining the active/inactive status of each exception we only need to keep track of whether we are still on the path. In other words, we treat all left links as wildcards during the traversal and hence do not cull the exception if we walk off the particular branch it specifies. Rather, as long as its current field is ‘0’ then traversing to the right keeps us on the path and as long as its current field is non-zero then traversing to the left keeps us on the path regardless of which instance in the pattern we are traversing. When the node location for the modification is reached, we can determine what to do to each OCTOR path based on the modification information and the on/off state. If the component at the path destination target is being deleted, we can delete the exception and optionally warn the designer. If we are off the path or changing the occurrence count, the path does not need to be changed. If we are deleting a node, the path field corresponding to this node (actually, its link) is deleted. If we are inserting a node, we insert `0' when path field corresponding to the insertion node position is ‘0’ and ‘-1’ (wildcard) otherwise (implying that the path follows the node to the left).

6.6 Applications and extensions

The OCTOR representation and approach described has been illustrated using scene graphs, a general description for geometric compositions with widespread application including geometric modeling approaches such as CSG, BReps, and parametric models. For instance, it can be used to support modeling (e.g. representation, identification, and selection) of CSG primitives (e.g. a pattern of holes) and features in BRep models (e.g.

features in features) and parametric models. It can also be used for specifying and handling exceptions in animation design (e.g. choreography) and specifying reference sets for constraint satisfying transforms.

We identify several opportunities for future work.

(1) Extend OCTOR to support ranges. For example, a range is required to select all of the seats in the fifth through eighteenth rows of an airplane seating arrangement. A wildcard generalizes a field to the whole domain, while a range is a sequential subset of a domain. Ranges can be supported by using two OCTOR strings, thus we would like to explore approaches for specifying ranges. In addition, a scheme to support more complex selections such as periodicity (e.g. every n-th instance), diagonal selections, and alternating selections such as a checker pattern could be incorporated into a more general selection system (Figure 6.16).

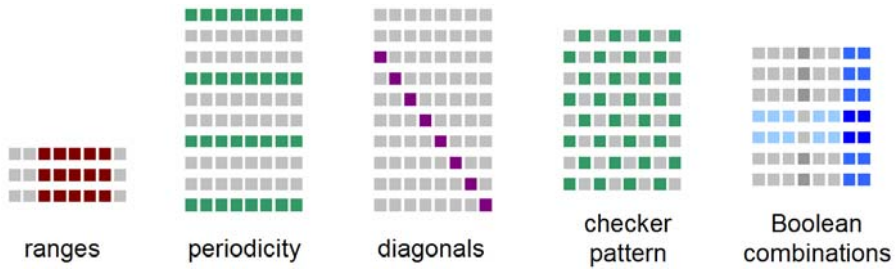

Figure 6.16: Possible extensions for a more general selection system include ranges, periodicity, diagonals, a checker pattern, and Boolean combinations.

(2) Develop a way to specify, represent, and compute Boolean combinations of OCTOR selections. This will greatly expand the set of possible selections one can make. Selections of Boolean combinations could also be incorporated into a more general selection system.

(3) Perform a user study on the selection approach and compare with other MOS approaches. The study would also assess the relative merits of the different selection guides.

6.7 Contributions and conclusions

In this chapter, we have presented an approach for modeling exceptions in recursive regular pattern hierarchies. In particular, OCTOR provides a representation for a subset of occurrences of modeling components and support for a user to graphically specify such subsets. The proposed representation based on a path and wildcards is more compact than a list of paths and is more general than a node with partial path. The set of subsets directly supported by the proposed scheme reflect the underlying designer intent contained in the assembly graph representation. The GUI support is flexible and offers several facilities for specifying selection sets. A 2-click method for making any OCTOR selection is described, but other methods including a k-click approach could be used. Refinement sets and other highlighting guides which aid the user in making the desired selection were also proposed. While the approach is well suited and intuitive for direct manipulation (direct picking), it also supports indirect picking of components, e.g. on an alternative visual representation such as a hierarchy outline. OCTOR provides a selection method that is less laborious than clicking each occurrence and less demanding on the user than specifying a partial path, not requiring knowledge of the underlying graph structure.

PART II

7 PROBLEM STATEMENT AND MOTIVATION

In PART I we discussed an approach and representation to designing and editing patterns of geometric features. A pattern is essentially a configuration of frames irrespective of the actual features and types of features instantiated with respect to the frames. In PART II we will discuss how to design and edit patterns of a specific kind of geometric feature: relief features on surfaces.

Relief feature processing has applications in ceramics, packaging, embossing (e.g. notaries and stamps), art, and architecture and reliefs can be executed on many different mediums including stone, marble, wood, stucco, terra cotta, porcelain, ivory, clay, metal, glass, ice, leather, paper, foam, and plastic (Figure 7.1, Figure 7.2, Figure 7.3, Figure 7.4, Figure 7.5, and Figure 7.6). Various names have been given to describe the different kinds of geometric reliefs which occur primarily in art and architecture. These include bas-relief (low relief) [Weyrich et al. 2007], semi-relief (medium relief), alto-relief (high relief), sunken relief, and hollow relief. Whether they are high or low in magnitude or raised, depressed, or both in direction, the common characteristic of these various kinds of reliefs is that they represent shapes that lie on or are cut out from some underlying surface. We say that relief features follow the surface or are *on* the surface.

Figure 7.1: These bottles illustrate reliefs executed on glass applied to packaging.

Figure 7.2: The bas-relief on Stone Mountain in Georgia is carved out of a quartz monzonite rock.

Figure 7.3: A metal keychain with reliefs.

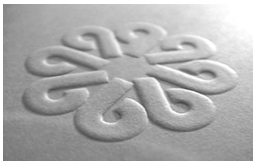

Figure 7.4: This embossing on paper illustrates a pattern of reliefs.

Figure 7.5: A ceramic teapot, mug, sugar pot, and milk jug are decorated with the same relief logo.

Figure 7.6: A set of flexible operations for relief feature transfer can aid the design of foam packaging.

In order to apply the concept of regular patterns to geometric relief features on surfaces, there are two main issues that need to be addressed, the first related to instantiating shapes (relief features) and the second to defining frames (on surfaces). To

define pattern regularity with respect to geometric content of features, i.e. a feature regular pattern, we need to be able to define a pattern leader as the unique feature content and a way to reproduce or duplicate its content via instantiation. To define pattern regularity with respect to spatial arrangement of frames, i.e. a frame regular pattern, we need a way to define a unique transformation by which subsequent frames in a series can be defined. Since we are concerned with relief features on surfaces in PART II, we need to define both feature instantiation and frame arrangement for relief features on surfaces. In this thesis, we make contributions related to relief feature instantiation and use a simple approach to arranging frames on surfaces which links regular arrangements in PART I to regular arrangements in PART II. We leave a full exploration of regular arrangements of frames on surfaces as future work.

The main problem we consider is concerned with feature content (as opposed to feature arrangement). In PART I we defined an instance as a shape (or a geometric modeling component in general) associated with a frame. However, in order to create patterns of relief features on surfaces, merely defining frames is not sufficient. To instantiate a relief feature, we need a clear definition of what the relief feature is and an approach for extracting and reapplying (cutting and pasting) a relief feature. This contrasts with a standalone instance in a scene graph which is trivially instantiated, given a frame. A relief feature is in some sense embedded on or in the surface and may not have a clear delineation of boundary or even content. Hence, we address the problem of how to extract and reapply a relief feature. In addressing this problem, which we call the *relief feature transfer* problem, we propose a practical definition for relief features. Then,

based on this definition we describe an approach that enables us to copy, paste, cut, delete, move, and slide relief features on surfaces.

Given this set of basic single-feature operations, we suggest how to use them to define operations for multiple relief features and in particular regular patterns of relief features on surfaces. These operations include *recognition* (for identifying patterns of relief features on surfaces), *feature regularization* (also known as beautification or homogenization) of pattern instances (for feature regularity), propagated editing of pattern instances (*modification*), and *exceptions* (with respect to feature regularity). Then, given an approach to define regular arrangements of frames, we additionally define the following editing operations for regular patterns of relief features on surfaces: *frame regularization* to homogenize the pattern transform (for frame regularity), re-spacing (adjusting the pattern transform), changing the pattern count, and transform exceptions. Note that these operations on multiple relief features fit in the context of the overall modeling schema as set forth at the outset of this thesis. While we do not present solutions to every one of these problems, we define these operations with respect to the set of basic single-feature operations on surface reliefs. The basic operations for regular patterns of relief features on surfaces are summarized in Figure 7.7.

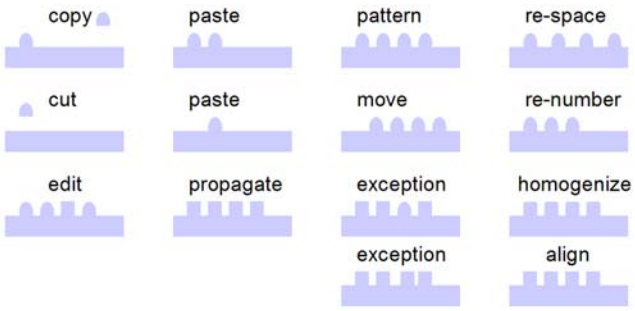

Figure 7.7: A set of basic single-feature operations and regular pattern operations for relief features on surfaces.

Our specific contributions are as follows:

- We propose a practical definition for relief features.
- Based on this definition we describe an approach that enables us to copy, paste, cut, delete, move, and slide relief features on surfaces.
    - Specifically, we propose a new interpretation for a 2D rectangular image, which we call an *imprint*, which can be used as a representation to transfer geometric relief features from one surface to another.
    - Based on imprints, we present the *imprint-mapping* approach to extract, process, and transfer relief features from one surface to another.
- Based on our imprint-mapping approach, we present a set of algorithms to copy, paste, cut, delete, move, and slide relief features on surfaces represented as triangle meshes without re-sampling or modifying the connectivity.
- Given an identified set of relief features on a surface, we describe how to edit the set as a regular pattern of relief features – in particular, feature regularization, relief feature editing, and feature exceptions. We describe how a basic set of imprint editing components can be combined to support these operations on multiple relief features on surfaces.

The rest of PART II is organized as follows. Chapter 8 gives a presentation of the relief feature transfer problem and a review of the related prior art on relief and detail transfer. In Chapter 9 we describe our imprint-mapping approach to relief feature transfer and provide algorithms to copy, paste, cut, delete, move, and slide relief features on surfaces. We also describe how to edit patterns of relief features with our approach. In

Chapter 10 we describe our prototype implementation of our relief feature editing approach and give results. Finally, the conclusion chapter summarizes the overall pattern editing framework presented, lists our specific contributions within the framework, and identifies areas for future work according to the framework.

8 RELIEF AND DETAIL TRANSFER

We review approaches to transferring geometric shape from one surface to another surface. More specifically, the problem we consider is how to extract geometric relief and detail information from a portion of a source surface and transfer that geometric shape information to modify a portion on a target surface (Figure 8.1). As a way to support relief feature cut and paste operations, removal of relief features is also discussed.

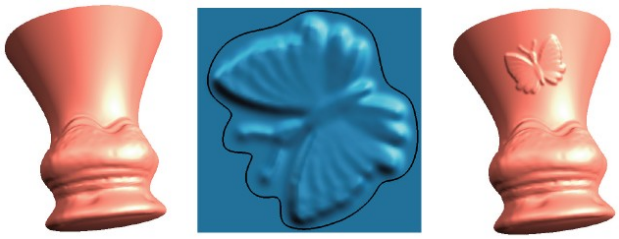

Figure 8.1: Relief feature transfer (from [Biermann et al. 2002] Figure 12).

The problem of relief and detail transfer is different from cut and paste editing of surfaces in general. In general cut and paste editing of surfaces, specifying a boundary on the source surface is sufficient to define a feature to cut. In other words, the portion or surface *chunk* itself is the content we want to transfer and the source chunk replaces the target chunk or fills a target hole. Hence, we refer to this idea as *chunk-based* transfer and the kind of feature to be transferred as a *chunk feature* (Figure 8.2). However, in relief and detail transfer, defining a boundary merely indicates the surface region containing the feature. The relief and detail information which represents a *relief feature* needs to be extracted from the region of interest of the surface. Then this relief and detail information,

not the actual chunk itself containing it, is used to change the shape of a region of a target surface on which we wish to paste the relief feature, not replace it.

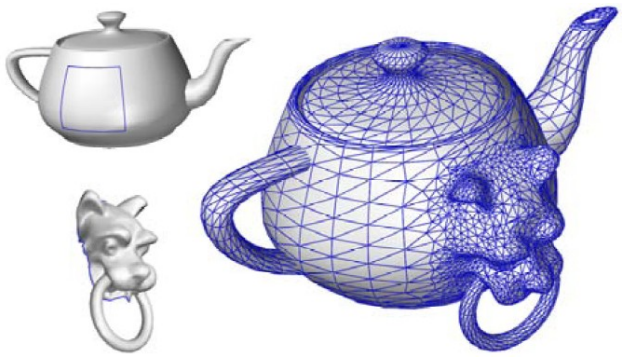

Figure 8.2: Chunk-based cut and paste (from [Yu et al. 2004] Figure 11).

We now seek a general definition for a relief feature. The prior art gives a variety of interpretations to what a relief feature is. [Biermann et al. 2002] define a detail surface as the difference between the actual surface and the base surface. [Sorkine et al. 2004] use the term "coating" to refer to the high-frequency surface details, or more specifically, to the difference between the original surface and a low frequency band of the surface. [Liu et al. 2006] define a geometric relief as extra material added (or subtracted) locally to some underlying surface. They note that the added material forms a surface with sculpted features clearly different from the underlying or surrounding surface. For the relief to be distinguishable, they further assume that the relief is raised or embossed at a small height relative to the features on the underlying surface. Similar to [Biermann et al. 2002] and [Sorkine et al. 2004], [Kolomenkin et al. 2009] view reliefs as the composition of a smooth base surface and a height function defined over that base. In their interpretation of reliefs, [Zatzarinni et al. 2009] additionally propose that the curvature of the base surface be smaller or equal to the curvature of the actual surface. The differences in these

interpretations are made evident in the details of the different approaches, which we will later discuss. However, from the prior art we can conclude that the basic idea that defines a relief feature is the existence of an underlying base surface on which the relief or detail lies. Hence, while specifying a closed curve on a surface is sufficient to define a chunk feature as a subset of the surface, additional processing is required to extract a relief feature from a given subset of the surface, i.e. to locally "separate" the relief feature from the base surface.

Now we would like to understand how this difference in interpretation affects which techniques are more or less appropriate to handle the transfer of specific surface features. In particular, there is no single approach to extract relief and detail information that fits all situations. The appropriate approach depends on the designer's interpretation of what a relief feature is. For example, if the designer wishes to reproduce the exact shape of a portion of a surface, e.g. a screw hole needs to retain its shape since its dimensions are defined to fit a screw, an approach which extracts just the finer details, e.g. only the threads of the screw hole, would not be appropriate. Rather, a chunk-based approach which can trivially perfectly reproduce the overall shape of a portion of a surface by simply clipping the hole at its boundary, placing the exact duplicate somewhere else, and "stitching" the boundary "seam" would be more appropriate (Figure 8.3bfj). Hence, we may say that a chunk-based approach naturally supports this kind of transfer by definition.

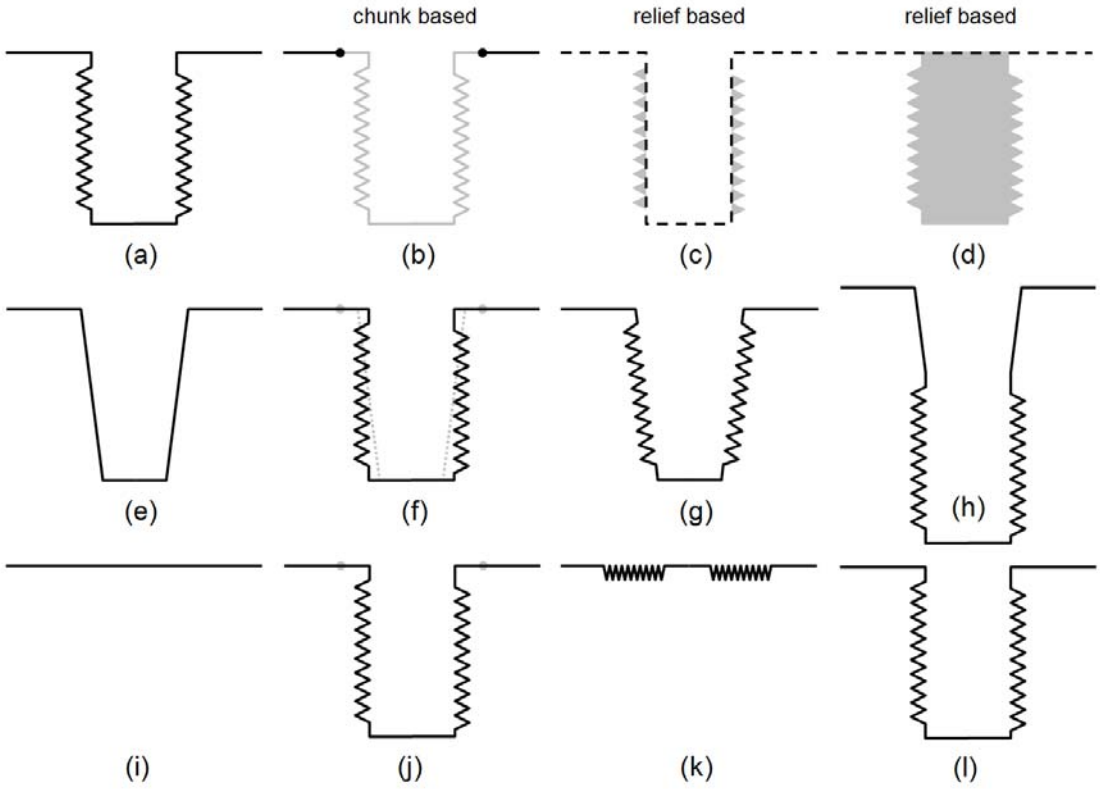

Figure 8.3: Comparison of chunk-based transfer (b/f/j) and relief-based transfer (c/d/g/h/k/l) with two different choices of base surface (c/d). The original source surface is given in (a) and the features copied in (b/c/d). The different target surfaces are given in (e) and (i) and the results of pasting given different types of transfer are given in (f/g/h) and (j/k/l), respectively. With the appropriate choice of base surface, a relief-based approach can obtain identical results to a chunk-based approach, e.g. (f/j) versus (i).

In contrast, a relief-based approach naturally lends itself to peeling the textured bumps off of a theoretically smooth underlying surface and applying it onto another smooth surface different in shape resulting in a surface with similar details but different overall shape. For example, the threads extracted from a cylindrical screw hole can be applied to a conical hole to form a conical screw hole (Figure 8.3cg). At the same time however, it is possible to achieve the same results as chunk-based transfer with a relief-based approach [Ma et al. 2006]. For instance, by appropriately defining the base surface, e.g. as a reference plane (Figure 8.3d), a relief-based approach can achieve perfect

reproduction of overall shape including details, e.g. the screw hole with its threads (Figure 8.3i). Hence, we also make certain assumptions about the kinds of features we aim to support and design an approach to handle such features. By making these assumptions, we intentionally limit the class of features supported in favor of allowing for an approach which has certain computational (simplicity and efficiency) and functional (relief signal processing) benefits which we will describe in more detail later in this thesis.

With this basic and intentionally general understanding of what a relief feature is, we explore ways of transferring geometric relief features from one surface to another. We organize our literature review as follows. We first define our notation and terminology. We then list the basic steps in a generic surface feature transfer approach. (Note that this framework supports the idea of both chunk-based and relief-based feature transfer.) Here we list the main issues and show how they fit into the overall feature transfer process. Then we discuss each problem one by one, explaining how the prior art addresses the main issues. Finally, we compare some of the most relevant prior art to the problem of relief feature transfer and organize them according to their capabilities and affordances. This will later help us situate our work and contributions in the context of the prior art.

8.1 Terminology and notation

Based on the wide variety of possible definitions and interpretations for a surface feature, there are a wide variety of proposed solutions to the problem of relief and detail transfer. Hence, it is difficult to find a single set of terms and notation which both succinctly and comprehensively describes all the elements in all possible approaches. Here we distill some of the basic elements common to many previously published

approaches which are useful in describing the essence of the relief and detail transfer problem.

Let M^S denote the source surface that contains a source feature region F^S which contains the feature of interest. Hence F^S is a simply connected proper subset of M^S. Let D^S define the boundary of F^S in M^S. (Note that in general D^S may have more than one component, e.g. to cut the handle off of a mug. One may also consider the multiple letters of an embossed word to be part of one feature, though in this case we may also consider them as separate relief features without loss of generality.) Likewise, let M^T be the target surface. Let F^T be the region of M^T that will be affected by the feature pasting operation. Hence F^T is a proper subset of M^T. Let D^T define the boundary of F^T in M^T. (See Figure 8.4.)

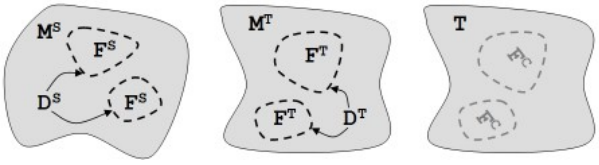

Figure 8.4: Source surface (left), target surface (middle), result of transferring F to target surface (right).

Let B^S be the base surface corresponding to the region of M^S that is bounded by D^S. That is, B^S is the base surface of F^S and has the same boundary as F^S. B^S can be used as a *replacement surface* to replace F^S in a delete or cut operation and it can be used as a *reference surface* to extract relief feature information from F^S. Likewise, let B^T be the base surface of F^T which is the region of M^T that is bounded by D^T. Hence, B^T has the same boundary as F^T. (See Figure 8.5.)

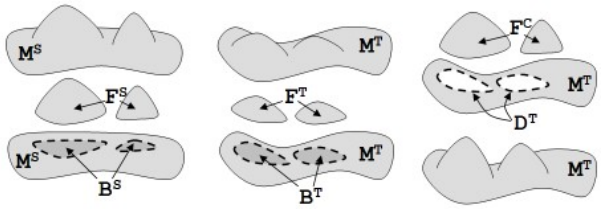

Figure 8.5: Feature regions F^S and F^T are replaced with corresponding base surfaces B^S and B^T at both the source and target (left and middle). A composed feature F^C derived from F^S and optionally F^T is pasted on the target surface M^T.

Before F^S can be pasted (added or attached to the target surface M^T), it may necessary to modify its shape it to help ensure continuity at the boundary after pasting. This may involve warping or reshaping using the shape of F^T or B^T. We refer to this modified feature as a composed feature F^C and its boundary as D^C.

Certain relief transfer algorithms may make use of a *context region* C^S surrounding F^S. Specifically, let C^S be the region of M^S with interior boundary D^S and exterior boundary E^S. Note that the interior boundary D^S is shared by F^S and C^S. Likewise, let C^T be the region of M^T with interior boundary D^T and exterior boundary E^T. C^T is the context region of F^T. (See Figure 8.6.)

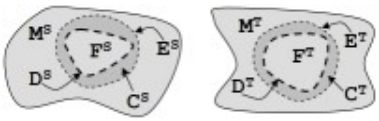

Figure 8.6: Context regions C^S and C^T are subsets of M^S and M^T that are bounded by D^S and D^T on the inside and E^S and E^T on the outside, respectively.

Given this notation, we can describe a set of unary editing operations for relief features; that is, each operates on a single feature. Here we summarize them.

Get (copy): Given D^S, a *get* operation obtains feature representation R^S from F^S and B^S.

Remove (delete): A *remove* operation replaces F^S with B^S.

Cut: A *cut* operation is the get operation followed by the remove operation.

Put (paste): Given D^T and R^S, a *put* operation replaces F^T with F^C, a composite of F^S and F^T which is F^S modified to fit D^T. This may involve computing R^T from F^T and B^T.

Move: A move operation is a cut followed by a put.

Slide: The slide operation is a series of one or more small moves.

8.2 Relief transfer process

We now discuss the main issues in the problem of relief and detail transfer.

The basic process of relief feature cut and paste can be summarized with the following basic steps (Figure 8.7). First, the user or feature detection software identifies a region F^S containing the source feature by specifying or computing its boundary D^S on source surface M^S (SELECTION). The goal of SELECTION is to identify a subset of the surface as the chunk feature or as the region of interest containing the relief feature. Then the system extracts the source feature into representation R^S. For a chunk feature, the representation R^S can have the same representation as the surface M^S and the boundary D^S serves to identify the relevant portion of M^S. For a relief feature, a local base surface B^S in the selected source region is computed (BASE COMPUTATION) and then the feature is encoded with respect to the base (ENCODING). The base surface B^S is generally a smooth, flat, or nearly flat surface homeomorphic to a disk which has D^S as its boundary, but could also be a smooth or flat surface which approximates D^S or F^S. The feature R^S is then encoded as the difference between the original surface F^S and the base surface B^S. Then the user identifies the target region F^T on target surface M^T where the feature will be pasted by specifying boundary D^T on M^T (TARGET SELECTION). The result of target selection is a closed curve D^T on M^T, some indication of which side of the

curve ("inside/outside") contains the feature F^T, and a correspondence between D^S and D^T. Next, the system composes the source feature R^S (optionally together with R^T, which is likewise encoded from F^T and B^T) into a feature to be pasted R^C (COMPOSITION). Composition allows the source feature to be blended with the surface at the target, which can be used, for example, to help ensure continuity at the boundary after pasting. Hence, composition often requires a map or correspondence to be established between the source and target features. Finally, the system decodes or reconstructs the feature to be pasted F^C from encoded representation R^C (RECONSTRUCTION) and pastes the feature on the target region (PASTING). Reconstruction generates the connectivity (if necessary) and computes geometric shape information of the final pasted feature in its final pose. Pasting attaches the connectivity of the reconstructed feature ("stitching" or "zipping") and performs additional smoothing at the transition boundary between the pasted feature and the target surface.

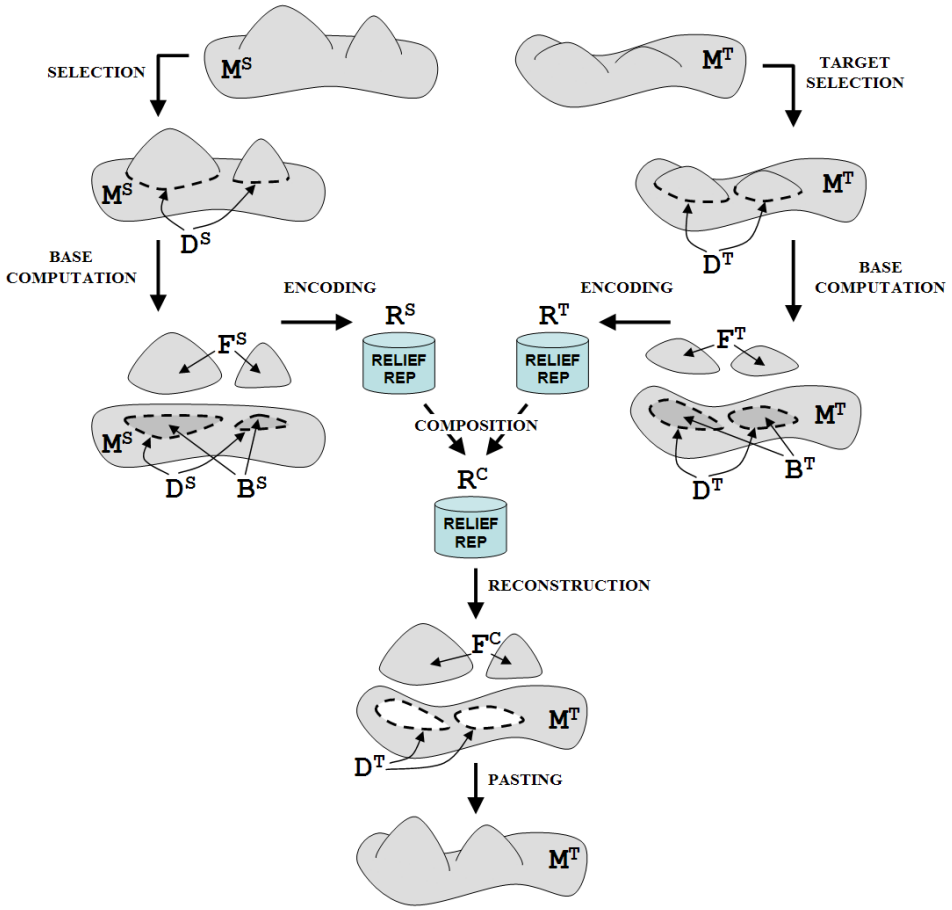

Figure 8.7: Surface feature cut and paste pipeline.

This process and notation is mainly derived from the work of [Biermann et al. 2002] and [Fu et al. 2004] but it also incorporates ideas from [Masuda et al. 2004] and [Ma et al. 2006]. Note that the way the process is denominated may or may not precisely fit or be perfectly consistent with all approaches, but we present it as a general guiding framework for the overall idea of geometric relief feature transfer on surfaces. We now describe these steps in more detail, showing how the prior art addresses each problem. Our goal is not to provide a comprehensive survey of every possible approach to solving every problem in the process. Rather, we focus on approaches of specific relevance to the solution we propose in this thesis. Hence, the problems of SELECTION, TARGET

SELECTION, BASE COMPUTATION, and PASTING will be covered in a more general way and the problems of ENCODING, COMPOSITION, and RECONSTRUCTION will be covered by describing specific approaches in the prior art.

8.2.1 Selection

In order to copy or cut a feature out of a surface, the user first needs to indicate which region of the surface to cut out or which contains the feature. This problem, which we call source selection or simply *selection*, is the process through which the designer selects F^S in M^S. The result of this process is the boundary D^S on M^S which separates F^S as a subset of M^S. To achieve this is to perform a segmentation of the surface. For simplicity, we will focus our discussion on selecting a region F^S that can be segmented with a single boundary. The process of tracing D^S accurately to reflect the designer's objective is often tedious; hence, we will discuss various attempts at automating or facilitating the process.

Many approaches have been proposed which solve the segmentation problem. We may roughly classify them as global optimization approaches or local search approaches [Masuda et al. 2004].

One approach to source selection is to compute a global segmentation and then have the user select one of the segmented regions as the portion to be cut [Attene et al. 2008]. There are numerous global segmentation approaches, some of which are surveyed by [Attene et al. 2006] and [Shamir 2008]. These approaches segment by focusing on some distinguishing characteristic such as extreme feature points, tubular shapes, feature scale, or fitting primitives. In general, no single automatic segmentation technique is always better for all types of objects [Chen et al. 2009], so it may be left up to the user to select the appropriate one to achieve the desired segmentation. Even so, approaches have been

proposed which are specifically designed for relief segmentation. [Zatzarinni et al. 2009] uses relief analysis to compute a global segmentation. They compute a height function over the entire surface and use a Gaussian mixture model to define a threshold to derive iso-contours on the surface which serve as possible segmentation boundaries.

A local segmentation approach provides a way for the user to directly specify a precise cutting boundary. This can be achieved by manually drawing strokes or placing points, segments, or curves to indicate the cut boundary. Manual segmentations have been found to be remarkably consistent, that is, people tend to segment objects in the same ways [Chen et al. 2009]. However, a precise boundary, e.g. at a surface crease, may be tedious or error-prone to obtain manually, whether using a single stroke or multiple strokes and whether using one view or multiple views. Because of this, user-guided cuts have been proposed. The intelligent scissoring approach to user-guided cuts lets the user make a cut which the system adjusts [Lee et. al. 2004; Funkhouser et al. 2004]. A similar approach lets the user paint a region and the system makes the cut within that region. Masuda et al. [2004] and Zelinka and Garland [2006] describe such approaches which use the graph cuts formalism for mesh segmentation proposed by Katz and Tal [2003]; that is, a graph defined on the connectivity of a mesh (e.g. graph nodes at faces and graph edges between faces corresponding to their mutually adjacent edge) with weights assigned based on surface characteristics (e.g. the angle at the edge between two faces). Liu et al. [2006] describe how to perform local segmentation of isolated reliefs using user-initialized snakes (an active contour model [Kass et al. 1988]) which are evolved and adapted to conform to the boundary of the isolated relief. Liu et al. [2007a] adapt this approach to segment reliefs from textured background surfaces. These many approaches

give the user a way to select single isolated surface features without being burdened with specifying the exact details of the boundary which can be tedious, time-consuming, and error-prone.

Overall, there is a vast body of prior art which addresses the basic problem of surface segmentation. For the purposes of selection in the context of relief cut and paste, we simply note that it is possible to present the user a variety of alternatives or find one that is appropriate for the application.

8.2.2 Target selection

The second problem, which we call *target selection*, is the process through which the designer selects F^T in M^T. The result of this process is the boundary D^T and an indication of which side of the boundary represents F^T, e.g. via orientation of the curve on the surface using a default convention. It is typically desirable for D^T to reflect the source feature F^S to be pasted or, in particular, its boundary D^S. Hence, we may rephrase the target selection problem as the process through which the user specifies where the feature F^S should be pasted on M^T, the result of which is the boundary D^T and a point-to-point correspondence between D^T and D^S.

In general, we would like to determine D^T directly based on F^S or D^S so that it is compatible with the feature we want to paste. We say that D^T and D^S are *compatible* if there exists an affinity such that $D^T=D^S$. The difficulty here lies in the fact that there is in general no rigid transformation or even affinity that places D^S on M^T. That is, a curve D^T on M^T that is compatible with D^S may not exist and, hence, the shape of D^S may be incompatible with M^T. Hence, most approaches focus on finding D^T on M^T that is

approximately compatible with D^S and establishing a correspondence between D^S and D^T and in some cases a correspondence between F^S and F^T.

To address the target selection problem, some approaches compute D^T from D^S by computing a cross-parameterization (a shared parameterization) between the surfaces on which they lie, i.e. between M^S and M^T, respectively; that is, both M^S and M^T are locally parameterized to the same parametric domain [Biermann et al. 2002; Fu et al. 2004]. For example, Biermann et al. [2002] encode D^S, map this to a planar parametric domain P^S with respect to the source surface, which is then mapped to a planar parametric domain P^T with respect to the target surface, which is then mapped back to the target surface M^T to obtain D^T. Biermann et al. propose to parametrically encode the source feature boundary D^S using a center point and a set of geodesic lines emanating radially from the center point to the boundary. The user picks a point on the surface F^S representing the approximate center (or the centroid of the boundary of the feature may be used) and specifies an orientation (e.g. by specifying an angle between the starting points of the parameterized source boundary and target boundary to be computed). The user can optionally specify a spine as a list of points instead of a single point which can support a lower distortion parameterization (encoding) of the boundary in the case of features with long and thin boundaries. (Note that the spine is more general than a single point and it includes the single center point.) This spine information is mapped to a local planar parameterization P^S of M^S in the neighborhood of the spine containing F^S. (Biermann acknowledges that this is a chicken-and-egg problem, but find that only an approximate region where the feature will fit is need.) Assume that the points $c_0,\ldots c_{m-1}$ are the spine points in the parametric domain P^S. A discrete parameterization of D^S is represented as a

list of triples, each triple consisting of the spine point c_i, the distance d_j from c_i to a boundary point w_j, and the angle γ_j of the direction from c_i to w_j and the spine. Now on the target surface, given a user-specified initial position and orientation for a point on the spine, Biermann computes the boundary D^T by performing geodesic walks out from the spine according to the list of triples. Because the geodesic walks are performed in the planar parametric domain, the resulting computed curve is guaranteed to close.

Fu et al. [2004] take a similar approach to Biermann et al. [2002] for determining D^T from D^S. Instead of walking around a center point or spine, they walk the outer boundary directly using distances and turning angles. First, the user is allowed to pick a starting point on the boundary and an orientation on D^S (which is on M^S) and a corresponding starting point and orientation on M^T. Given these starting points and orientations, Fu computes D^T by performing a geodesic walk of D^S on M^T. This is achieved by walking around the loop D^S in a planar parameterization P^S of M^S, which maps to planar parameterization P^T of M^T. In other words, P^S and P^T are mapped to be a common parameterization between M^S and M^T and this common parameterization is used to map points on M^S to points on M^T. Hence, a loop on M^S can be mapped to a loop on M^T via this common parameterization. A closed loop on P^S maps to a closed loop on P^T since angles and distances are preserved in the plane. The points on the loop are then mapped from P^T back to M^T resulting in D^T. The main difference between this approach and [Biermann et al. 2002] is that the approach of Fu et al. walks the boundary directly using turning angles (Figure 8.8b) while the approach of Biermann et al. walks the boundary by walking from a center point using angles with respect to this center point (Figure 8.8a).

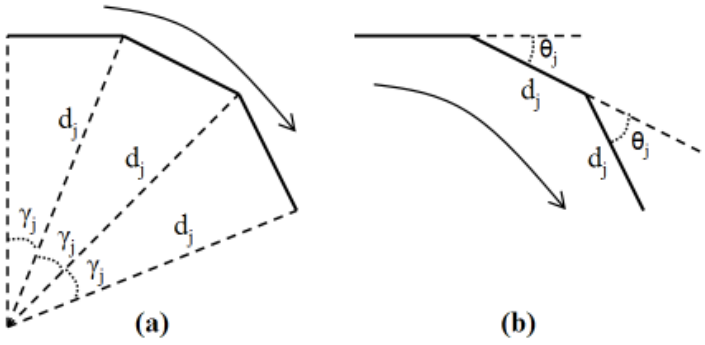

Figure 8.8: Two ways of walking around a curve: (a) use angles and distances from a center point and (2) use turn angles and walk distances.

Some approaches simplify the D^S to D^T compatibility problem by explicitly allowing F^S to be warped [Kanai et al. 1999; Masuda et al. 2004; Yu et al. 2004]. These approaches then rely on obtaining and using a correspondence between D^S and D^T and in some cases between F^S and F^T to facilitate the warp. For example, Kanai et al. [1999] requires the designer to select a sparse and equal number of vertices in sequence on both M^S and M^T. The sequences are sparse, so they may be connected with geodesic edges to form a more detailed boundary or using an interpolating spline to obtain a smooth boundary. These vertices are ordered and the sequence has an orientation with respect to the surface (clockwise or counter-clockwise); hence, the correspondence between D^S and D^T and the occupancy (which side of the curve is the interior or exterior of F^T) is effectively given by the designer. Based on the boundary correspondence, an interior mapping between F^S and F^T is computed and used to warp F^S such that D^S is compatible with D^T.

Masuda et al. [2004] fit a parametric b-spline volume around feature region F^S such that the bottom of the volume approximates a user-specified context region C^S around F^S. This fitted b-spline volume defines a volumetric parameterization of F^S (including D^S).

The parametric b-spline volume is then fit to the target surface such that the bottom of the volume approximates a user-specified target region F^T (or optionally its context region C^T). These two volumetric parameterizations and fittings, one at the source and one at the target region, define a warp of F^S into F^C (the feature to be pasted) and a transformation to place F^C on the target surface. M^T is then re-meshed to include D^C (the boundary of F^C); hence, the fitted and warped D^S becomes D^T.

Yu et al. [2004] project the source boundary D^S directly onto target surface M^T in a user-defined direction to obtain D^T. Hence, the correspondence between D^S and D^T are given by construction.

In certain cases, it may be appropriate to achieve target selection with the same approach as source selection. For instance, when the designer wants to specifically constrain the target boundary D^T to be a certain specified shape, the designer may be free to specify an arbitrary boundary D^T along with a point-to-point correspondence between D^S and D^T. This correspondence can be established using a user-specified sparse number of points on both the source and target surface as in [Kanai et al. 1999] or using a user-specified starting point and direction and walking the boundary similar to [Fu et al. 2004]. That is, given sparse correspondences or a beginning point and a direction, the complete correspondence between D^S and D^T can be computed by parameterizing the boundary curves by length along the curve.

To summarize, there exist a variety of possible approaches to target selection. Some approaches directly compute the target boundary D^T from the source boundary D^S [Biermann et al. 2002; Fu et al. 2004; Masuda et al. 2004; Yu et al. 2004] and hence, the correspondence between D^S and D^T is given by construction. Other approaches allow for

more arbitrary D^T and hence, an explicit correspondence needs to be established as a separate step. This correspondence computation is typically user-guided, using a sparse number of corresponding points and computing parameterizations for each curve to define the complete correspondence. Overall, we suggest that there is not one single approach that is best for all situations; the appropriate choice depends on the kind of features to be transferred and the way they are extracted, encoded, and pasted as well as the design task and user skill level.

8.2.3 Base computation

Given the boundary D^S of feature region F^S on a surface M^S, the process of *base computation* computes a base surface B^S. Given our assumption that D^S is a single closed loop, our goal is to compute a surface B^S that is homeomorphic to a disk and either includes or approximates D^S as its boundary. (Note that while our discussion uses the terms D^S, F^S, M^S, and B^S, our discussion on base computation also applies to the target region using the terms D^T, F^T, M^T, and B^T, respectively.)

Before discussing approaches to base computation, we first distinguish two functions for a base surface. Firstly, a base surface B^S can be used as a *replacement surface* to replace feature region F^S which can be used to facilitate a cut or remove operation. In this case, B^S should be computed to respect certain constraints at the boundary D^S of region F^S. Hence, in addition to D^S, base computation approaches may also make use of a context region C^S when computing B^S [Masuda et al. 2004]. C^S can be loosely defined as a region on M^S that is adjacent to D^S and outside of F^S. In other words, C^S surrounds F^S and D^S and may provide useful context information for computing the base surface B^S. For example, D^S alone provides positional constraints for the boundary of B^S, but C^S can

be used to provide normal or curvature constraints for filling the hole bounded by D^S (Figure 8.9).

Figure 8.9: Filling a gap with D^S only (top) versus filling with D^S and C^S (bottom). In this case, the context region provides normal constraints in addition to positional constraints at the boundary.

Secondly, a base surface B^S can be used as a *reference surface* (c.f. [Barghiel et al. 1995]) for extracting and encoding the relief feature in F^S into a transferable representation R^S. In other words, a base surface can be used as a reference surface with respect to which a relief feature is defined. A reference surface that is smooth or flat distinguishes itself from relief feature detail which has higher frequency content [Biermann et al. 2002; Sorkine et al. 2004]. In this sense a smooth or flat surface is intrinsically better suited to be a reference surface than one that contains many high frequency perturbations. In contrast, for a replacement surface it is not necessarily preferred to have any particular characteristic such as being smooth or flat, but a particular application, task, or user preference may call for a replacement surface that maintains coherence or aesthetics across D^S [Sharf et al. 2004]. For example, if the region surrounding D^S on M^S is smooth then it may be more appropriate for the replacement surface to be smooth rather than bumpy; if the region surrounding D^S on M^S is bumpy then it may be more appropriate for the replacement surface to be bumpy rather than smooth.

In our discussion of approaches for base computation, we emphasize approaches for computing reference surfaces, though many of the approaches also generate valid replacement surfaces. Hence, we focus our discussion on approaches which aim to compute a **smooth** surface patch (as opposed to prioritizing aesthetics for example) that meets **or approximates** the boundary constraints of the hole defined by D^S.

Approaches to computing a smooth base surface can be roughly classified as being based on smoothing, interpolation, or fitting [Biermann et al. 2002]. (1) Smoothing removes high frequency details from the surface F^S resulting in a surface B^S that is a smoothened version of F^S. (2) Interpolation-based approaches compute the surface to fill a hole by interpolating the boundary D^S. (3) Fitting or variational approaches attempt to fit a surface such that it approximates the boundary constraints at D^S, the feature region F^S, or the context region C^S. Hence, a fitting approach can be used to compute a reference surface while avoiding the potential complexities of having to satisfy the boundary constraints. (Note that in practice, a base computation approach may make use of or apply concepts from more than one of these three techniques [Kobbelt 2000].) The technique of smoothing typically operates on existing geometry, e.g. on F^S which already spans D^S, while interpolation and fitting are often used in the context of hole-filling approaches. Hence, we will first discuss smoothing approaches and then discuss interpolation and fitting approaches together in the context of hole-filling and surface completion.

8.2.3.1 Smoothing

Smoothing approaches can be used to determine B^S by modifying F^S. In particular, Taubin [1995] showed that the formalism of signal processing could be applied to

geometry processing. Thus, surface smoothing can be accomplished via a low pass filter. (For coverage of more recent approaches, we direct the reader to the literature surveys found in [Vallet and Levy 2008], [Kim and Rossignac 2005], [Pauly and Gross 2001], [Tasdizen et al. 2002], and [Mokhtarian et al. 1998].) Of specific interest to us, a typical smoothing approach removes high frequency details from an existing surface by iteratively applying a local fairing operation over the whole region to be smoothed. In the context of relief feature cut and paste, the region F^S already spans the hole with boundary D^S. Thus if F^S is homeomorphic to a disk, it can be smoothed to obtain B^S. If F^S is not homeomorphic to a disk, F^S is removed and the hole defined by D^S needs to be filled using a hole filling approach (discussed later) [Barequet and Sharir 1995; Curless and Levoy 1996; Held 2001]; then a smoothing approach can be applied to obtain B^S [Liepa 2003].

Rather than using geometric information from only D^S or C^S as a pure hole filling does, a smoothing approach also uses information from the contents of F^S when computing B^S. A smoothing approach can also use information from C^S to provide additional constraints, e.g. to enforce normal constraints at the boundary D^S. This requires the smoothing approach to consider a large enough neighborhood to incorporate information from C^S. For example, a bi-Laplacian approach [Schneider et al. 2001] uses weights from a 2-ring neighborhood for computing each point and thus is able to factor in tangency at D^S while a Laplacian approach using weights from a 1-ring neighborhood would only be able to factor in positional constraints at D^S.

Smoothing approaches can be used to produce a continuum of possible B^S with varying frequency content (Figure 8.10). For example, Laplacian smoothing on a triangle

mesh is achieved by perturbing each vertex v towards the average of its 1-ring neighborhood [Sorkine et al. 2004]. Iteratively applying this local fairing operation to perturb all of the vertices in F^S generates a continuum of increasingly smooth B^S.

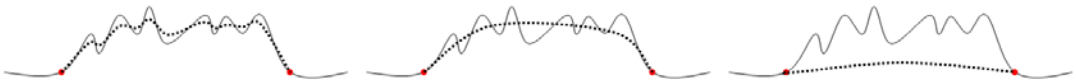

Figure 8.10: Base surfaces computed from various levels of smoothing.

8.2.3.2 Hole-filling and surface completion

A base surface B^S can be computed by considering D^S as the boundary of a hole and then computing a spanning surface to fill the hole. There are two main components to this process: (1) generating the connectivity to span the hole and (2) computing the geometry. These two aspects are typically executed concurrently and are inseparably linked in hole-filling algorithms. However, it is also possible to compute the mesh connectivity to span the hole first and then evaluate an interpolant (in an interpolation based approach), e.g. [Farbman et al. 2009], or minimize a functional (in a fitting based approach) to determine the geometric positions of the spanning surface connectivity, e.g. [Biermann et al. 2002].

The problem of hole filling is well-explored in the prior art and the solutions are diverse. Many early approaches have been proposed in the context of surface reconstruction [Bajaj et al. 1995; Curless and Levoy 1996]. Triangulation approaches can easily handle small holes and gaps and can also be effective in spanning planar or nearly planar holes [Barequet and Sharir 1995; Held 2001]. For filling holes which are geometrically complex, volumetric approaches have been shown to be successful [Curless and Levoy 1996; Sharf et al. 2004; Davis et al. 2002]. The basic idea is to

compute and use a volumetric field such as a distance field (e.g. using distances to the existing surface) to propagate the surface out from the boundary. This propagation can be carried out in a coarse-to-fine manner. For further exploration of the topic, we direct the reader to the literature surveys in [Held 2001], [Sharf et al. 2004], [Davis et al. 2002], and [Branch et al. 2006].

8.2.3.3 Transfinite interpolation

Interpolation can be used to generate a base surface B^S by computing geometry to fill or span the hole defined by the boundary D^S. In particular, transfinite interpolation is the construction of a function over a planar domain that matches a given function on the boundary [Dyken and Floater 2009]. To use transfinite interpolation for the purpose of filling a hole defined by 3D boundary curve D^S, D^S needs to be redefined as a closed, non-self-intersecting 2D curve G^S and a scalar height function h^S defined on the domain of G^S. This can be achieved by defining a plane (A, N), where A is a point on the plane and N is the plane normal, and projecting D^S onto the plane. Then the height function can be computed as $h^S(i)=AG^S(i)\bullet N$, where $h^S(i)$ is the height function value corresponding to point $G^S(i)$ on curve G^S. Then, the height function value for any point on the planar domain within G^S can be computed using a transfinite interpolant. For arbitrary curves in 3D, it is not always possible to compute the closed curve G^S without self intersections, e.g. a knot; however, in the case of relief features, D^S may often be defined as a 2D curve projected in some direction N. Hence, the same direction N could be used to convert D^S into G^S and h^S.

In the context of 2D image editing, hole-filling algorithms based on transfinite interpolation have been employed to compute a membrane surface for the seamless

transfer of image clips (a portion of an image contained within a closed curve defined on the image domain) [Perez et al. 2003; Farbman et al. 2009]. A membrane surface represents how the discrepancies at the boundary between the image clip to be pasted and the target image are spread out into the interior of the image clip to be pasted. A membrane which smoothly spreads out the error can be computed by solving a Poisson equation with Dirichlet boundary conditions. Such a membrane can also be computed using a harmonic (or membrane) interpolant [Farbman et al. 2009]. While these approaches define functions having a planar domain and color values as the range, i.e. three (e.g. red, green, and blue) scalar functions, the 2D interpolation problem can also be interpreted as a 3D hole-filling problem by defining the scalar function at the boundary to represent the positions of the boundary points as a height field. Hence, the spanning surface is computed by evaluating an interpolant with respect to the boundary values for each pixel in the interior of G^S. Many interpolants are available for this purpose including harmonic coordinates [Joshi et al. 2007], mean value coordinates [Floater 2003; Ju et al. 2005; Dyken and Floater 2009], Green coordinates [Lipman et al. 2008], and complex barycentric coordinates [Weber et al. 2009]. In such an image grid based approach, evaluating the interpolants may be computationally expensive, especially when evaluated at every pixel. To address this, Farbman et al. [2009] propose to compute an adaptive triangulation (with fine pixel-scale edge lengths at the boundary and larger edges in the interior) in the plane and only evaluate the interpolants at the vertices. The values for the pixels inside the faces of this triangulation are then interpolated from the values at the vertices. In this way, a 2D approach can be used to efficiently generate a 3D spanning surface B^S.

8.2.3.4 Fitting

Fitting can also be used to compute a surface which fills or approximately fills the hole defined by boundary D^S. This is typically achieved by computing the parameters of a specific shape representation that minimize a functional. For example, compute the normal and offset of a plane that minimizes the sum of square distances of a set of points, e.g. from the boundary D^S, to the plane. Hence, a base surface computed using fitting would not be a suitable replacement surface if the surface does not satisfy the boundary constraints, but can still be used as a reference surface.

Many fitting approaches attempt to fill the hole by fitting a smooth surface patch that meets (or approximately meets) the boundary constraints of the hole [Ilic and Fua 2003; Verdera et al. 2003; Pernot et al. 2003b; Song et al. 2004]. The approaches are diverse and a full treatment of the topic is beyond the scope of this thesis. Instead, we mention a few to give the reader an idea of the range and scope of approaches and refer the reader to the literature reviews in [Litke et al. 2001], [Savchenko and Kojekine 2002], and [Ilic and Fua 2006] for a more comprehensive exploration of the topic.

Biermann et al. [2002] use a least-squares fitting approach to compute the control points of a multiresolution subdivision surface at coarser levels. Specifically, for each subdivision level, they find the control points such that minimizes the sum of squares distance between the vertices of the original mesh and the smooth surface obtained by subdividing the surface given those control points to the finest level. A continuum of base surface choices is then available via a single parameter which selects the level in the multiresolution hierarchy. [Litke et al. 2001] base their approach to fitting subdivision surfaces on the method of quasi-interpolation. [Pernot et al. 2003b] use a physics-inspired

simulation to enforce shape constraints at the boundary. In particular, they use a mechanical model to minimize the curvature variation across the boundary. Other surface fitting approaches which have been successfully used to fill holes include fitting an implicit surface to a mesh [Ilic and Fua 2003], variational implicit function fitting based on the normal vector field [Verdera et al. 2003], and finite element energy minimization (e.g. Wilmore energy) [Clarenz et al. 2004].

Fitting may also be used to obtain a reference surface while avoiding the potential complexities of having to satisfy the boundary constraints. Consider the example of fitting a plane (which is flat) to the boundary (which may not be flat). In this example, fitting computes a surface which approximately covers the hole defined by boundary D^S but does not necessarily seal the boundary. Hence, the resulting surface would not be a suitable replacement surface (which needs to satisfy the boundary constraints) but can be used as a reference surface. Strictly speaking, a reference surface is not required to satisfy the exact boundary constraints at D^S. It only needs to satisfy the requirements of the feature encoding approach (discussed later), e.g. every point on F^S can be specified as an offset of a point on B^S. With this goal in mind, a fitted approximation of F^S itself (as opposed to just its boundary) could be considered a valid reference surface (Figure 8.11).

Figure 8.11: A plane fitted to the boundary (left) versus a plane fitted to the feature (right). Both methods compute valid reference surfaces. Fitting to the boundary generates a more appropriate replacement surface, though in general it may not be possible to avoid gaps at the boundary.

Finally, some approaches do not require the surface patch to be smooth but still attempts to meet the boundary constraints. For instance, [Sharf et al. 2004] and [Savchenko and Kojekine 2002] use fitting to compute how the boundary of a surface patch should be warped to approximately conform to the boundary of the hole. The patch is warped and a subsequent smoothing step is applied smooth the connection at the boundary. The result is a patch that can contain arbitrary geometry in the interior but still approximately meets the constraints at the boundary. Such a surface would then be more appropriately used as a replacement surface rather than as a reference surface, particularly if the patch interior is not smooth.

We have seen that there are a plethora of tools available for the purpose of base computation. In particular, approaches exist which can fill a hole with a smooth surface, fit prescribed surface patches over holes to fill them, and ensure smoothness or satisfy other requirements at the boundary. These approaches provide the ability to compute replacement base surfaces which can be used to facilitate relief feature cut and delete operations. In addition, fitting approaches can be employed to compute "free-floating" surfaces. Such surfaces may not be appropriate for use as replacement surfaces but could be used as reference surfaces, that is, surfaces used for the purpose of extracting/encoding and applying/pasting relief features. Hence, reference surfaces are not necessarily required to meet the boundary constraints unless the encoding and pasting approach requires it. Next, we explore such approaches.

8.2.4 Encoding, composition, and reconstruction

Here we discuss the problems of encoding, composition, and reconstruction. These three processes describe how shape information is transferred from the source to the target and are intrinsically linked. Hence, we discuss these three problems together.

Let us first define the three problems. Given feature region F^S and base B^S (or in particular a reference surface), the process of *encoding* computes a representation R^S which encodes the relief feature information contained in F^S. The purpose of encoding is to extract relief feature information into a representation that can be used to transfer the relief feature to another surface. The expectation is that R^S can be used to recover F^S from B^S. The target feature region F^T may also be encoded as R^T for the purposes of composition.

Given R^S and R^T, the process of *composition* computes R^C, a composition of the source feature to be pasted (represented as R^S) with the target feature existing at the target location (represented as R^T). The primary purpose of composition is to facilitate the flow of relief feature information from the source to the target. Composition also provides a means for the existing surface detail at F^T to influence the final shape of the pasted feature. This may be useful for smoothly transitioning between existing detail at the target (at the boundary) and the pasted detail (moving inward from the boundary) or mixing the relief information to create novel shape and detail [Sorkine et al. 2004; Biermann et al. 2002; Zatzarinni et al. 2009].

Given feature representation R^C, *reconstruction* is used to produce F^C. This process essentially decodes the encoded representation R^C back into the surface feature representation F^C and thus can be considered the inverse of encoding. Hence, an ideal

reconstruction reproduces F^S from R^S. The result is that after cutting a feature and pasting it back, the original surface is recovered.

When considering how to transfer geometric relief information, it is useful to distinguish two components of the shape information: the connectivity (e.g. the edges and faces of a polygonal mesh) and the geometry (e.g. the positions of the vertices of a polygonal mesh). Given connectivity information for both the source and target, a mapping between the two is often needed for facilitating the transfer of the geometry information. Hence, before discussing geometry transfer, we discuss ways of computing such a mapping.

8.2.4.1 Connectivity transfer

We first consider the transfer of connectivity. When the topology of the target region F^T is not homeomorphic to the topology of the source region F^S, the connectivity of the target region must be modified or replaced before the geometry information is transferred. For example, if F^S contains a handle and F^T is homeomorphic to a disc, we must alter or replace F^T with connectivity that has the topology of a handle. Chunk-based transfer achieves this by replacing F^T with F^S (or F^C) hence, both the geometry and the connectivity are transferred [Fu et al. 2004; Ma et al. 2006]. In a relief-based transfer approach, the connectivity needs to be prepared to accept the geometric relief detail. This can be achieved using hole filling, surface completion, or some other base computation approach for computing replacement surfaces (which we have already discussed). The main requirements for the connectivity are that the topology (e.g. the genus) is the same and the sampling is sufficiently dense to articulate the transferred source detail. Once the connectivity is computed, it is ready to be shaped using the geometry information.

8.2.4.2 Mapping source to target

While a chunk-based approach passes the geometry information with the connectivity, a relief-based approach needs to compute or establish a mapping between the geometry information on the source and the target regions since the source and target connectivity is typically different. This mapping allows the connectivity (e.g. the vertices) of the target to receive the correct geometry information to form the shape for reconstruction. In other words, the connectivity provides the modeling medium and the mapping overlays the shaping "blueprint" which contains the geometry information. In a chunk-based approach, the geometry values are already mapped to the connectivity and no further computation is required to reconstruct the surface representation for pasting. In this sense, chunk-based approaches are more appropriately suited to transferring geometry of arbitrary topology.

The process of mapping the source surface to the target surface for the purpose of transferring relief information may involve computing a mapping between F^S and F^T or between B^S and B^T. (Note that in the case of chunk-based transfer, computing a mapping between the boundaries D^S and D^T is sufficient since connectivity and geometry are transferred together. However, if a warp is required, a mapping between the interior may sometimes be needed to facilitate the warp as discussed later.) This mapping can be direct between the source and target (e.g. vertex to vertex) or indirect by computing a cross parameterization between these source-target pairs.

Direct mappings. Biermann et al. [2002] note that in the case of a direct mapping between the source region to the target surface domain, the values of the mapping are not part of any affine space. For instance, to identify a point on a triangle mesh, we would need a triple (i, u, v), where i is the triangle id and (u, v) are coordinates within the

triangle. Thus, without reparameterization there is no simple way to compute linear combinations of two arbitrary points, e.g. for interpolating values when the connectivity is not identical, which is the typical case. Furthermore, for cutting and pasting a single feature multiple times in multiple locations, a new correspondence and mapping must be established for each target location. Hence, as many other approaches use an indirect mapping, we focus our discussion on computing indirect mappings between the source and target surfaces.

Indirect mappings. An indirect mapping maps both the source and target feature regions to a common intermediate domain, typically a planar domain. Such a coupled mapping is called a cross parameterization. A cross parameterization is computed by parameterizing both the source and target to parametric domains P^S and P^T respectively. The parametric domains can then be easily mapped to each other, e.g. with an affinity. Then, values, i.e. shape information, can then be passed from the source to the target using interpolation to obtain information between samples [Biermann et al. 2002; Sorkine et al. 2004; Zatzarinni et al. 2009].

Parameterization. For our purposes, it may suffice to say that many parameterization techniques exist which compute one-to-one and onto mappings directly between surfaces which aim to minimize distortion, many of which compute planar parameterizations (a one-to-one mapping from all the points on a surface to a planar domain), the result of which can be used define cross parameterizations. Nevertheless, for completeness we briefly discuss parameterization. Parameterization techniques are important in many geometric modeling and texturing applications and a variety of algorithms have been proposed including general parameterization methods [Floater 1997]

for reparameterization (i.e. changing connectivity to semi-regular) [Eck et al. 1995; Guskov et al. 2000; Krishnamurthy and Levoy 1996; Lee et al. 1998] and texture mapping [Levy 2001; Praun et al. 2000; Maillot et al. 1993; Zhang et al. 2005]. For a more comprehensive discussion of parameterization techniques, we refer the reader to the parameterization survey by Sheffer et al. [2006].

In particular, many possible approaches are suitable for establishing mappings for the purpose of surface feature extraction and pasting. Here we mention a few which have been successfully employed in prior work. The approach of [Sheffer and deSturler 2000] to parameterize a surface over a plane with low distortion using triangulation flattening has been applied by [Biermann et al. 2002] to encode, transfer, and paste surface features. [Kuriyama and Kaneko 1999] embed the vertices of a mesh in a normalized two-dimensional space which can be controlled by a lattice of control points for deformation. [Masuda et al. 2004] fit a b-spline volume to the surface region containing the feature to be cut. The b-spline volume, which provides a volumetric parameterization for the feature, is fitted and deformed to the target surface region to facilitate pasting. This illustrates that parameterizations other than planar ones can be used to transfer surface feature information.

8.2.4.3 Geometery transfer

We now consider transfer of geometry information that describes the shape of the feature to be transferred. Given that we know how to compute mappings between the source surface and target surface or between both to a common domain, we then need a way to encode shape information with respect to the mapping that allows shape information to be passed from the source surface to the target surface.

There are a variety of possible encoding approaches. To help us better understand the range of existing approaches, we classify encoding approaches by the kind of reference frame they use to encode the geometry of the surface. More specifically, we classify encoding approaches as intrinsic or extrinsic. Intrinsic encodings are based on local frames intrinsic to the surface element being encoded, e.g. based on surface normals corresponding to the vertex being encoded [Kobbelt et al. 1999; Lee et al. 2000; Biermann et al. 2002; Zatzarinni et al. 2009] or on Laplacians [Sorkine et al. 2004]. Extrinsic encodings encode surface elements with respect to a "less local" frame, typically a unique frame for the whole feature, e.g. in a chunk-based approach [Kanai et al. 1999; Ma et al. 2006], or even a globally-defined frame, e.g. a single arbitrary or user supplied direction [Szymczak et al. 2002].

We now discuss the various encoding approaches one by one. We focus on prior art which describes encoding geometry information for the purpose of feature transfer. Since composition and decoding are closely related to the discussion of the encoding representation, we will also discuss them here.

8.2.4.3.1 Intrinsic encodings

[Biermann et al. 2002] use a local coordinate system consisting of the base surface normal and two tangent vectors (corresponding to two partial derivatives of the surface parameterization which they compute). The detail surface is then defined as a triple (d^n, d^{t1}, d^{t2}), which can be thought of as a scalar displacement d^n along the normal and a tangential displacement (d^{t1}, d^{t2}) in parametric coordinates. Given original surface F(x) and base surface B(x), the detail surface can be defined as D(x) = F(x) – B(x), where x is a point on the parametric domain that corresponds to both F (x) and B(x) and hence maps

them to each other. The local frame on the base at B(x) is defined as (n_B, $\partial_1 B$, $\partial_2 B$), where n_B is the normal at B(x) and dB = ($\partial_1 B$, $\partial_2 B$) can be thought of as map which maps vectors in the plane to vectors in the tangent plane of the surface. Hence, the detail surface can be decomposed into the scalar displacement d^n along the normal n_b and a tangential displacement in parametric coordinates $d^t = (d^{t1}, d^{t2})$. Thus, the equation relating the original surface, the base, and the details can be written as: $F(x) = B(x) + dB(x)d^t(x) + n_b(x)d^n(x)$.

Note that both source and target surfaces are separated into base and detail parts. Now let P be a map from B^S to B^T, which defines how the surface is pasted. The result of pasting is a new surface F^C with the same base as F^T but with the details taken from F^S. We may state this as $F^C = B^T + (dB^T\ dP\ d^{tS} + n_B{}^T\ d^{nS}) \circ P^{-1}$, where all functions are evaluated at a point x^T in the region of parametric domain P^T corresponding to B^T. The symbol $\circ$ denotes function composition; hence, the composition of differentials $dB^T \circ dP$ is used to transform the tangential component of details. This establishes the mapping between the local frames on the source and target surfaces. Now details from the source and target surfaces can be mixed during pasting simply by using a weighted average of the scalar displacements d^{nS} and d^{nT}.

[Fu et al. 2004] store detail vectors between the vertex on the feature surface F and the point on the base surface B to which it has been mapped via cross-parameterization. Their encoding approach allows them to encode and transfer features with arbitrary topology as long as a parameterization can be established between the connectivity of the feature and the base. Instead of using the normal vector at B(x) as the direction with which a scalar displacement d^n is defined, they use a cross parameterization (specifically,

the intrinsic surface parameterization of [Desbrun et al. 2002]) to link F(x) to B(x). Specifically, let v^F be a vertex in F that is mapped to a point w^B on B. The point w^B can be found by first mapping v^F to point v^{FP} in the parametric domain P^F, then finding a face f^{BP} containing v^{FP} in P^B, and then finding w^B on face f^B of B via the barycentric coordinates of v^{FP} on face f^{BP}. Finally, the local detail vector $\mathbf{d}^F = v^F - w^B$ is obtained and associated with the base point w^B and the base face f^B containing w^B.

To transfer the feature to the target region, the base surface is used as a detail carrier. B^S is mapped to B^T using a cross parameterization and then the geometry of F^C is the reconstructed on B^T as if it was B^S. Details are not provided in [Fu et al. 2004], but we assume that the vertex v^{TF} can be reconstructed by finding $\mathbf{d}^F$ in the local frame defined by w^B and f^B and then reconstructing $\mathbf{d}^{TF}$ using w^{TB} and f^{TB}. For example, we could establish a consistent convention to pick the triangle normal N, obtain a vector E_1 from one of its edges, and compute mutually orthogonal vector $E_2 = N \times E_1$. The encoded coordinate would then be $V = (x,y,z)^T = M\mathbf{d}^F$, where M is the matrix $(E_1, E_2, N)^T$. (Here ^{T} means matrix transpose.) In other words, we use three dot products to obtain the x, y, and z coordinates of V, respectively. Then, we can compute E_1, E_2, and N from the target face f^{TB} and compute $\mathbf{d}^{TF} = x \cdot E_1 + y \cdot E_2 + z \cdot N$ and then $v^{TF} = w^{TB} + \mathbf{d}^{TF}$.

[Sorkine et al. 2004] uses Laplacian coordinates to encode each vertex v with respect to a local frame centered at the average a_v of its neighboring vertices. The Laplacian coordinate, which is a differential coordinate, is thus computed as a vector $L_v = v - a_v$. For the purpose of transferring higher frequency surface detail from a smoother version of the surface (which they call "coating transfer"), they generate the base B using smoothing (e.g. [Desbrun et al. 1999]) and encode the coating at vertex v as $R_v = L_v - L_w$,

where w is the vertex on the base corresponding to v and $L_w = w - a_w$ is its Laplacian coordinate. Note that the encoded detail vector R_v is the difference between the Laplacian coordinates which are vectors. These detail vectors can be computed for both the source and target surfaces F^S and F^T, respectively. Then, in order to transfer the detail information to a vertex v^T in the target region, a cross parameterization between B^S and B^T is first computed and then the detail vectors associated with the source surface are sampled to obtain a detail vectors from which to reconstruct the new surface F^C at the target. There are many ways to do this, but Sorkine et al. find that first mapping the 1-ring of v^S (using the cross parameterization) and then computing the Laplacian from this mapped 1-ring yields good results. They note that such an approach assumes locally similar distortion in the mapping. A simpler alternative is to linearly interpolate the three Laplacian coordinates sampled at the triangle vertices to which v^T maps. While this approach leads to some "blurring" compared with the first approach, it is simpler and avoids the parametric distortion problem. Additional care is taken to make sure the tesselations or sampling density of the source and target surfaces is "compatible" or at least locally fine enough to accommodate the detail of the other mesh. They achieve this through local, isotropic remeshing [Alliez et al. 2003; Surazhsky and Gotsman 2003; Vorsatz et al. 2003]. Finally, the reconstructed surface exhibiting the coating transfer from F^S to F^T is given by $F^C = L^{-1}(\Delta + R^S)$, where L^{-1} is the transformation from Laplacian coordinates to absolute coordinates, Δ denotes the Laplacian coordinates of the vertices of F^T (i.e. L_v for all v^T in F^T), and R^S are the encoded detail vectors of the source surface F^S with respect to B^S. Mixing of source and target detail vectors can be achieved

by weighted average of the corresponding detail vectors in R^S and R^T, the result of which yields R^C. Then, $F^C = L^{-1}(\Delta + R^C)$.

Another element of Laplacian-based encoding and reconstruction is the matter of reference frame. When a Laplacian coordinate L is transferred, the translation element is included but rotation and scaling are not transferred. Sorkine et al. hence describe how to explicitly apply rotation and isotropic scaling to the Laplacian coordinates. The basic idea is to define a local frame using the point a (average of the neighbors of v), the local normal, and another vector orthogonal to the normal (tangent to the surface), each of which are intrinsic to the surface. Then the transformation is applied which brings the source frame to the target frame. More details can be found in their paper [Sorkine et al. 2004].

[Zatzarinni et al. 2009] take a similar approach to [Biermann et al. 2002]. They encode surface detail at a vertex v as a scalar defined as a signed displacement from the corresponding point on the base surface in the direction of the base surface normal. The original surface F, the base surface B and the detail are related by the expression: $v^F = w^B + h(v^F) \cdot n^B(w^B)$, where v^F is a vertex on F, w^B is the point on B associated with v^F, $n^B(w^B)$ is the normal of surface B at w^B, and $h(v^F)$ is the signed displacement of v^F with respect to the base point w^B. Hence, the detail surface representation R = h(x) is essentially a height map defined over the base surface. To transfer the detail, they find a planar cross parameterization between F^S and F^T and pass heights via cross parameterization and interpolation as described by [Sorkine et al. 2004]. Then, given that w^{SF} is the point on F^S mapped to vertex v^{TF} on F^T and v^{TF} corresponds with point w^{TB} on B^T, the expression $v^{CF} = w^{TB} + h(w^{SF}) \cdot n^{TB}(w^{TB})$ is used to construct F^C by using the source surface displacement

$h(w^{SF})$ and the target base surface normals $n^{TB}(w^{TB})$. Mixing of source and target details is achieved using a weighted average of height values. This can be expressed as $v^{CF} = w^{TB} + (a \cdot h(w^{SF}) + b \cdot h(v^{TF})) \cdot n^{TB}(w^{TB})$. Note that v^{CF} is actually a height adjusted v^{TF}. Hence, the relief feature transfer is accomplished by warping the target surface F^T and no additional pasting procedure is required to join the pasted feature F^C to the target surface.

8.2.4.3.2 Extrinsic encodings

Other approaches, particularly chunk-based approaches, encode all of the geometry in a feature with respect to a single coordinate frame [Kanai et al. 1999]. The frame could be arbitrary or computed from the surface, e.g. a reference plane and its associated normal [Ma et al. 2006]. The source surface geometry, e.g. vertices, are then captured in this reference frame. Then, the feature is aligned to the target using a rigid transformation. This transformation can be specified manually [Kanai et al. 1999] or the system can guide the alignment, e.g. by snapping using fitting or registration of the base reference surfaces [Ma et al. 2006], the boundary, or the context region near the boundary [Sharf et al. 2006; Sorkine et al. 2004].

[Masuda et al. 2004] fit a parametric volume around the source feature F^S (as described in Section 8.2.2) and hence the 3D parameterization doubles as the 3D encoding. In other words, each point inside the volume has a parametric coordinate. (Determining such a parametric coordinate given a 3D point turns out to be non-trivial. We do not use, extend, or directly compete with their approach but mainly avoid such complexities altogether; hence, it is beyond the scope of this thesis and we refer the reader to the paper [Masuda et al. 2004] for more details.) As a result, to bend and warp the parametric volume is to bend and warp the space in which the feature is encoded. In

particular, they fit this parametric volume to the target surface, thus defining a warp. Using the parametric volume as the "detail carrier", they reconstruct the surface chunk F^S with respect to the parametric volume fitted to the target surface. Hence, the pasted feature F^C has the same connectivity as F^S but with geometry adjusted by rigid transformation and the warping of the parametric volume.

[Szymczak et al. 2002] encode geometry with respect to a set of axis-aligned rays. While they apply this encoding for the purpose of mesh compression, such an idea could also be applied to feature transfer.

8.2.4.4 Summary

We have discussed various approaches to encode, compose, and reconstruct relief features on surfaces. The approaches make use of a diversity of techniques including those for parameterization, fairing, fitting, and change of coordinates (transformation). While their technical details may vary, the basic idea of most of the approaches is to encode geometric detail information with respect to a frame of reference, which can be taken locally as an intrinsic frame, semi-locally to a larger portion of the feature, or the whole feature may use a single frame. Detail may be encoded as displacements in some set direction such as the surface normal or as vectors in the related frame. This information can be used to reconstruct the feature at the target by defining respective frames at the target and mapping the target to the source, which may be accomplished through cross parameterization or fitting and alignment. In approaches which deform the existing connectivity with the transferred detail, this completes the feature transfer. In other approaches, additional procedures may be required to attach the pasted feature

according to the requirements of the designer. We discuss these procedures and techniques next.

8.2.5 Pasting

The goal of *pasting* is to attach F^C to M^T given D^T. This assumes that the target region has already been determined and the feature F^C is already composed, reconstructed, and placed relative to D^T. In relief transfer approaches based on deforming the existing target surface, the pasting step is already accomplished. Hence, we mainly consider the pasting step as the set of procedures used to attach the connectivity at the boundary (previously referred to as "stitching" or "zipping" the "seam") and to smooth the transition geometrically. We will not discuss the user interface and interaction aspects of pasting as in [Chan et al. 1997].

First, we discuss attachment of connectivity. The need for connectivity attachment arises in several places in the feature transfer process. A replacement base surface computed to fill the hole defined by boundary D^S or D^T may need stitching if it was computed via fitting. Hence, attachment may be needed at both the source (for cut and delete) and the target (for replacing existing connectivity so that new feature information can be transferred). In our discussion on target selection, we described several approaches which compute D^T so that we could paste F^S possessing boundary D^S and pointed out that there is in general no rigid transformation or even affinity that places D^S on M^T. Even with warping, gaps may still exist. We can see this in the specific approaches. [Biermann et al. 2002] and [Fu et al. 2004] use a cross parameterization to transfer the boundary D^S onto M^T to compute D^T. [Ma et al. 2006] computes a reference plane at the source and target and aligns them to position feature to be pasted F^C. [Masuda et al. 2004] fits a

parametric volume around the source feature and then fits the parametric volume at the target. This induces both a transformation and a warp to F^S resulting in F^C. In all these cases, the boundary of the feature to be pasted F^C and the target surface M^T are not guaranteed to align perfectly not only because the warp is not ideal, but because the tessellation of the respective boundaries are incompatible, that is, not identical. Hence, some form of topological surgery or tweaking is required.

Topological repair given closely aligned boundaries is generally straightforward. For example, [Biermann et al. 2002] add the additional vertices to the target surface during the target determination process. [Fu et al. 2004] use a hole filling algorithm [Barequet and Sharir 1995] to seal the gap between D^S (or D^C) and D^T. [Ma et al. 2006] and [Masuda et al. 2004] remesh the target surface mesh so that the vertices and edges of D^C are included in the mesh. Then the faces in the interior of this new border D^T are removed and the feature mesh is connected to the target mesh.

Simply attaching the connectivity of the pasted feature to the target surface achieves C^0 continuity; however, the designer may want the shape at the boundary of the pasted or affected region to have higher continuity or at least to exhibit some form of geometric consistency. There are various ways of achieving this with the basic idea being to apply some form of smoothing or blending. For example, [Fu et al. 2004] apply a smoothing operation (e.g. [Kobbelt et al. 1998]) locally at the border. [Biermann et al. 2002] use an alpha mask to compute a local base surface as a blend of the original source surface M^S and a global base surface (as opposed to locally constrained to boundary D^S). Consequently, the detail vectors vanish as the sample approaches the boundary and the pasted feature will apply little or no displacement near or at the target boundary D^T.

[Kanai et al. 1999] allow the designer to control the blending of the pasted feature explicitly. They cross parameterize the source and target via a supermesh that has the combined graph structure of both F^S and F^T in the parametric domain. They then map this supermesh to a circle, effectively establishing a radial parameterization with domain value 0 at the boundary and 1 in the center. Then they allow the designer to design a fusion control function (FCF) f(s) which is used as a weight to control the blending via a weighted average of the source geometry and the target geometry: $v^C = f(s)v^S + (1 - f(s))v^T$, where v^C, v^S, and v^T are corresponding vertices of F^C, F^S, and F^T, respectively. Any relief transfer approach that supports mixing of details can also use a similar approach to control the blending, e.g. [Biermann et al. 2002] and [Zatzarinni et al. 2009].

We have described a set of techniques used to finish the pasting process. These include topological gap filling and repair techniques as well as geometric smoothing and blending techniques. The appropriate techniques to use depend on the specific affordances of the basic relief transfer approach used. For an approach which simply warps the existing connectivity, a simple blending at the boundary may be sufficient, which an approach which replaces a surface portion with new connectivity could require a combination of warping, topological repair, and some form of smoothing at the boundary.

8.3 Comparison of relief transfer prior art

We now organize and compare some of the prior art on relief feature transfer. Here we mainly select approaches which present complete solutions to the relief feature cut and paste problem or which describe significant portions of the overall relief feature transfer process. In some cases, we have chosen to include an approach because it

produces similar results. For example, we have included several chunk-based approaches because of their relevance or use of the feature-base concept.

The papers we compare are “Cut-and-Paste Editing of Multiresolution Surfaces” [Biermann et al. 2002], “Laplacian Surface Editing“ [Sorkine et al. 2004], “Relief Analysis and Extraction” [Zatzarinni et al. 2009], “Topology-free cut-and-paste editing over meshes“ [Fu et al. 2004], “Volume-Based Cut-and-Paste Editing For Early Design Phases” [Masuda et al. 2004], “Cut-and-Paste Editing over 3D Meshes” [Ma et al. 2006], “Interactive Mesh Fusion Based on Local 3D Metamorphosis“ [Kanai et al. 1999], “Mesh Editing with Poisson-Based Gradient Field Manipulation“ [Yu et al. 2004], and “Interactive mesh dragging with an adaptive remeshing technique” [Suzuki et al. 2000]. In addition, we include work on relief segmentation by Liu et al. [Liu et al. 2006; Liu et al. 2007a; Liu et al. 2007b] and work on surface pasting based on simulated displacement mapping by Barghiel, Chan, Tsang, Conrad, Ma, et al. [Barghiel et al. 1995; Chan et al. 1997; Tsang 1998; Conrad and Mann 1999; Ma and Mann 2001]. We have already covered details from each of these papers throughout our discussion on the relief feature transfer process. Here we primarily summarize each focusing on a specific list of criterion.

We use the following criterion to characterize the capabilities, limitations, and basic feature transfer methodologies or techniques used in each approach.

1 – Feature types supported, e.g. non-zero genus topology, overhangs, and local height field.

2 – Supports chunk-based transfer. Connectivity and geometry are extracted from a portion of the source surface and used replace a portion of the target surface.

3 – Supports relief-based transfer. Geometric information representing the difference between the source surface and an underlying base surface is extracted and used to change the shape at the target surface with respect to its own underlying base surface.

4 – Performs relief feature removal (delete) and hence supports a cut operation (cut = copy + delete).

5 – Transfer methodology. This includes the method of encoding and mapping from source to target.

6 – Pasting methodology including target determination and reconstruction.

7 – Supports mixing source with target detail.

8 – Pasting connectivity, e.g. replace, adjust (e.g. multi-resolution adaption), preserve existing connectivity.

8.3.1 Biermann et al. [2002]

Biermann et al. [2002] present a comprehensive approach to copy and paste relief features on multiresolution surfaces. They describe how to compute a smooth base surface to different degrees of flatness controlled by a single user-selectable parameter. This base surface is used for separating the relief but can also be used as a replacement surface for deleting or cutting the feature. The relief feature details are encoded as a scalar displacement between the original surface and its base surface in the direction of the base normal. This encoding process can also be applied to the target surface region. The transfer of relief details is facilitated using a cross parameterization between the source and target surface regions. Hence, details of the source and the target can be blended. Their approach modifies the target connectivity using adaptive subdivision to ensure the target surface resolution can represent the pasted details to a sufficient fidelity.

In summation, the presented approach supports real-time transfer of relief features with overhangs but not handles, that is, features homeomorphic to a disc.

8.3.2 Sorkine et al. [2004]

Sorkine et al. [2004] use the term coating to refer to the high-frequency surface details, or more specifically, to the difference between the original surface and a low frequency band of the surface. This idea is related to work on multiresolution modeling [Guskov et al. 1999; Kobbelt et al. 1999]. To extract and apply these high frequency surface details, they use an intrinsic surface representation based on Laplacians. The essence of the approach is represent each point in a local Laplacian coordinate system. Smoothing can be performed by moving each point towards the origin with respect to their Laplacian coordinates and, hence, can be used to remove or delete relief details from a surface. Conversely, detail exaggeration is performed by moving it away from the origin. Analogously, the detail can be transferred to another surface by using or mixing in the Laplacian coordinates of the source at the target. Their approach is relatively straightforward if the source and target have the same connectivity. If the source and target have different connectivity, a map needs to be defined between the two surfaces. They compute this map as a cross parameterization, parameterizing the two surfaces over a common domain following mean-value coordinate parameterization [Floater 2003]. Hence, their relief feature transfer approach is able to preserve the connectivity of the target surface. The vertices of the target surface obtain the Laplacian-based offsets by interpolating values transferred via the cross parameterization.

Note that Sorkine et al. [2004] also describe a chunk-based surface pasting approach. The basic idea is to enforce constraints at the feature boundary and allow the rest of the

surface to be reconstructed using the intrinsic surface information encoded as Laplacians. They overlap the regions near the feature and target boundaries and perform blending of this transition region by computing a cross parameterization of the overlapped regions. They do not describe how the source feature or target region is determined or cut, how to position and align the feature, or how to compute a correspondence between the two cut seams in order to facilitate the cross mapping. This hints at the potential for chunk-based feature transfer approaches to be complex and require the fusion of many techniques, perhaps requiring custom or crafty engineering to implement in practice.

8.3.3 Zatzarinni et al. [2009]

Similar to Sorkine et al. [2004] and Biermann et al. [2002], Kolomenkin et al. [2009] view reliefs as the composition of a smooth base surface and a height function defined over that base. With this view in mind, Zatzarinni et al. [2009] present an approach for analyzing and extracting multiple general reliefs lying on general freeform surfaces. The approach finds a globally optimal base that minimizes the height function given an estimation of the base normals over a whole mesh surface. Technically, the approach does not actually need to compute the base surface; rather, the global solution to the height function gives the offsets from the base. The base itself can be derived from these offsets. Analysis is performed on this height function to compute a segmentation of the reliefs. A Gaussian mixture model is used to determine an iso-contour threshold to apply to the height function to determine the segmentation. A relief can be removed by suppressing the heights to zero resulting in the base surface.

Once the height function is computed for the source and the target surface, relief and detail transport is achieved by transferring offsets. Similar to Biermann et al. [2002] and

Sorkine et al. [2004], a map between the source and the target needs to be established. Once a map is defined between the two surfaces, the details at the target can be replaced by or mixed with the source details. The pasting process thus preserves the existing connectivity.

8.3.4 Fu et al. [2004]

Fu et al. [2004] present a relief transfer approach which supports features of non-zero genus. To achieve this, they first compute base surface passing through the boundary vertices of the selected feature region using a boundary triangulation technique. Then they encode feature detail based on a surface parameterization. Given a correspondence between the feature and base boundaries, an intrinsic parameterization [Desbrun et al. 2002] is computed to map the surface interiors. The result is that neighboring points are mapped to neighboring points while still supporting features with overhangs and non-zero genus. Then, the source base is attached to the target surface, replacing the target surface region. The feature is then reconstructed on the pasted base. Hence, the base surface is used as a detail carrier. Effectively, their approach is chunk-based when pasting the source base at the target, but relief-based when encoding and decoding the geometric details relative to the base surface. The base surface computed at the source feature region can be used as a replacement surface to support a relief feature delete or cut operation.

8.3.5 Masuda et al. [2004]

Masuda et al. [2004] present a chunk-based surface transfer approach with a volume based feature encoding. The main idea behind their approach is to use a spline volume as

the detail carrier. They first compute a feature region and a surrounding context region on the source surface based on user input. They compute a base surface based on the context region which is used to replace the feature region to support a delete or cut operation. To facilitate feature transfer, they fit a spline volume around the source feature such that the bottom surface of the volume approximates the base surface. The feature is parameterized in this spline volume. A spline volume is subsequently fit at the target region which defines a deformation of the feature to be pasted. The target region is then remeshed to include the boundary vertices of the pasted feature and the faces in this replaced target region are removed. The result is an approach which supports copy and paste of features with overhangs and non-zero genus and is able to avoid self-intersections of a deformed feature due to the volumetric parameterization.

8.3.6 Ma et al. [2006]

Ma et al. [2006] present a chunk-based approach for cutting and pasting triangle meshes. The approach uses least-squares fitting to the boundary vertices to compute a reference plane at both the source and target region. The reference plane not only facilitates the encoding of geometry information but the source plane is aligned to the target and facilitates the transformation of the source feature to the target pose. As in [Masuda et al. 2004], the target mesh is remeshed to include the boundary of the pasted mesh and the target region is removed, effectively replacing it with the pasted feature. The result is a surface cut and paste approach which supports non-zero genus features.

8.3.7 Kanai et al. [1999]

Kanai et al. [1999] present an approach to merge two meshes given the two mesh boundaries and a sparse vertex correspondence of the boundaries. Their approach can be applied to chunk-based surface pasting by considering one mesh as the feature and the other as the target. Their contribution is an approach to facilitate control over warping the feature using a user-definable blending function f(s) which controls the blend between the feature and the target surface region. They require both the source feature and the target region to be homeomorphic to a disc and compute a pseudo-radial parameterization from the boundaries (s=0) to the center (s=1) for each. These parameterizations are mapped to each other and used to compute a "supermesh" with the combined graph structure of the source and target surface regions. The positions of the pasted feature are then computed as a blend between the target geometry and the source geometry (after a rigid transformation into pasting position) using f(s) as the weight.

8.3.8 Yu et al. [2004]

Yu et al. [2004] present an approach that can merge two meshes given the two mesh boundaries and a vertex correspondence. The basis of their approach is a mesh solver based on the Poisson equation which modifies the mesh geometry implicitly through gradient field manipulation. The approach is hence able to enforce constraints at the boundary while spreading the error due to the boundary deformation out to the interior. This results in an approach which can merge meshes of arbitrary topology as long as a boundary correspondence can be established.

8.3.9 [Suzuki et al. 2000]

Suzuki et al. [2000] present an approach to dragging features on surfaces. A feature is defined by defining its boundary and hence can have overhangs and non-zero genus. Dragging is supported by adaptively remeshing at the feature boundary using local topological operations. The result is an approach which can move features from one place on a surface to another place on the same surface by small increments. The connectivity of the mesh on the dragging path is modified and the geometry will also be perturbed in the surface tangent direction and may be perturbed in the normal direction also if the surface is not perfectly flat.

8.3.10 Liu et al. [2006; 2007a; 2007b]

Liu et al. [2006] present an approach for segmenting reliefs from surfaces. They define a relief as extra material locally added to (or removed from) some underlying surface, making note that the added material forms a surface with sculpted features clearly different from the underlying or surrounding surface. For the relief to be distinguishable, they further assume that the relief is raised or embossed at a small height relative to the features on the underlying surface.

Based on their basic definition of a relief as well as their assumptions about the distinguishability of reliefs, [Liu et al. 2006] uses user-initialized snakes (or active contour model [Kass et al. 1988]) which are evolved and adapted to conform to the boundary of an isolated relief. This effectively computes a segmentation for a single surface patch. [Liu et al. 2007a] adapts this approach to segment reliefs from textured background surfaces. [Liu et al. 2007b] applies the approach to segmenting periodic reliefs.

Because the relief is segmented using a single exterior boundary cut, the approach is suitable for source determination, that is, in identifying the feature region to be copied or cut. A chunk-based approach is then immediately applicable, while a relief-based approach would require the computation of the underlying base surface to separate the relief.

8.3.11 Barghiel, Chan, Tsang, Conrad, Ma, et al. [1995; 1997; 1998; 1999; 2001]

Barghiel, Chan, Tsang, Conrad, Ma, et al. [1995; 1997; 1998; 1999; 2001] present approaches to pasting displacement reliefs on spline surfaces. They assume that the feature to be pasted and the target base surface is given and that both are represented by spline surfaces. The basis of their pasting approach is to simulate displacement mapping using a sparser set of spline surface control points. In doing so, they trade fidelity for speed. This enables their approach to run at real-time rates which makes it applicable to interactive pasting. Their approach can handle features describable as a displacement map and is well suited for smooth surfaces. Hence, a pasted feature with overhangs is not directly supported.

8.3.12 Comparison

Table 1 compares the relief transfer prior art we have summarized. Biermann et al. [2002], Sorkine et al. [2004], and Zatzarinnin et al. [2009] describe what we would call true relief-based surface feature transfer approaches. Fu et al. [2004] performs chunk-based transfer on the base but reconstructs the feature detail on the pasted base, treating it as a detail carrier. Masuda et al. [2004] and Ma et al. [2006] encode detail with respect to

a parametric volume and a plane, respectively, but these reference items are mainly used to define rigid transformation and warping (in the case of the parametric volume). They treat the features as surface chunks and clip and zip/stitch them at the boundaries accordingly. Similarly, Kanai et al. [1999] and Yu et al. [2004] use a boundary correspondence between the source feature and target to define a transformation and a warp, which facilitate quality joining of surface chunks at their boundaries. Suzuki et al. [2000] present a unique approach for transferring surface features using dragging. A chunk feature defined by a boundary can be slid across a surface to transfer its location on the surface. Chan et al. [1997] describe how to drag an offset feature across a surface to find a pasting location; however, they assume the feature and base are given and already separated as in Barghiel et al. [1995]. Finally, Liu et al. [2006] present a relief segmentation approach that can determine a relief boundary but does not perform relief analysis with respect to a base surface. Hence, it can be used in the source selection step to identify the region containing the relief feature, but separate analysis and processing would be needed to extract relief information within the identified region.

Table 1: Comparison of surface transfer approaches.

	Feature type	Chunk	Relief	Copy	Delete	Transfer (encode/ map)	Paste (geom)	Mixing	Paste (topo)
Biermann 2002	disc topo, overhangs		Y	Y	replacement	cross-param (over surface)	spine, radial geodesic walks, surface normal frames	Y	adaptive subdivision
Sorkine 2004 (chunk approach)	disc topo (handles, overhangs)	(Y)	Y	Y	detail supress	cross-param (over surface)	Laplacian reconstruction	Y	preserve
Zatzarinni 2009	height map (over surface)		Y	Y	height supress	cross-param (over surface)	offset deformation	Y	preserve
Fu 2004	handles, overhangs	topo	geom	Y	replacement	cross-param (feature-base)	geodesic boundary walk, surface param frames		replace
OUR APPROACH	height map (from a view)		Y	Y	height supress	single-view mapping	offset deformation	Y	preserve
Barghiel 1995	height map		Y				b-spline based deformation		
Masuda 2004	handles, overhangs	Y		Y	replacement	parametric volume	volume fit		replace
Ma 2006	handles, overhangs	Y		Y		plane mapping	plane fit	Y	replace
Kanai 1999	disc topo	Y		Y		boundary correspond	interior radial mapping	Y	replace
Yu 2004	handles, overhangs	Y		Y		boundary correspond	Poisson mesh solver		replace
Suzuki 2000	handles, overhangs	Y		move	local iterative replacement	local topo ops at boundary	slide		tweak
Liu 2006	handles, overhangs, distinguishable border	Y							

9 REPRESENTATION-INDEPENDENT RELIEF PROCESSING

9.1 Introduction

We present an approach to sliding relief features on freeform surfaces. In the context of freeform surface editing, the ability to grab part of the surface and slide it can be useful for tweaking the locations of features in the model. For instance, given a model of a bunny we may wish to slide its eyes an arbitrary distance forward towards the front of the face. As we are sliding, we may need to see the resulting model. Such real-time feedback can aid the designer in selecting the appropriate tweak to produce the desired result.

To achieve this we have three main challenges. Firstly, we need a way to determine what exactly the feature is. A closed curve on the surface can be used to define the boundary of the feature in the sense of a surface chunk, but a relief feature is defined as material added to or removed from an underlying base surface. Hence, a closed curve would merely serve to identify the region containing the feature. The relief feature itself needs to be extracted with respect to some base or reference surface. Secondly, as the feature is moved, we need to reconstruct the surface in the space from which the feature is moved. In other words, a replacement surface needs to be computed and placed or applied. Lastly, we need a way to reapply the feature in a new location on the surface. Given the extracted relief feature information and a target region, we modify the shape of the target region to reflect the relief feature information.

These three challenges can be summarized as a *relief feature transfer* (relief feature cut and paste) problem where the source and target surface regions are in close proximity

to each other (possibly overlapping) and the local neighborhoods are assumed to be homeomorphic to a disc. This relief feature transfer problem typically involves several steps, many of which can potentially involve non-trivial and even complex techniques and algorithms. (For more details, see the relief and detail transfer prior art in Chapter 8. An overview is given in Figure 8.7 of Section 8.2.) To summarize the process, first, a source and target region needs to be specified. This may involve the use of a segmentation algorithm to determine a precise boundary. Then, a relief-based approach may compute a base surface using a hole filling, surface completion, smoothing, or fitting approach. Then the feature is extracted and encoded with respect to the base surface, typically with respect to some local frames along the base surface or the local frames could be computed more directly from the source surface [Zatzarinni et al. 2009]. This may involve computing a reference base surface (for the purpose of encoding) separate from the replacement base surface (used for replacing the surface portion from which the feature was moved). A mapping, which could be direct or indirect, between the source and target is then computed. An indirect mapping involves parameterizing both the source and target surfaces (or their base surfaces) and aligning their parametric domains. Then the encoded shape information is passed from the source to the target and the target region is warped based on the encoded shape information. Reconstructing the shape at the target region may involve first replacing the target region with a base surface. Consequently, a pasting step involving "stiching" or "zipping" the boundary of the replacement into the hole vacated by the target region would be required.

Depending on the particular approach, many or most of these steps would need to be recomputed during each iteration of sliding the feature on the surface. This is an issue

since most of the steps, including target determination, base surface computation (for perhaps both reference and replacement surfaces separately), computing multiple mappings and parameterizations, local frame encoding and decoding, and pasting have the potential to be algorithmically cumbersome. Furthermore, in many approaches deleting and pasting a feature portion modifies the connectivity of the target surface. Repetitively deleting and pasting the feature in a slide operation can leave a trail of arbitrary connectivity, particularly when a hole filling approach is used to compute the replacement surface.

Suzuki et al. [2000] avoid the complexities of a general cut and paste scheme by designing an approach based on adaptive remeshing which is specifically tailored to interactive mesh dragging. Their adaptive remeshing evaluates the local deformation of faces induced by dragging and uses local toplogical operations to delete and split faces at the front and tail end of the advancing feature. Their approach preserves the existing mesh connectivity as the feature portion of the mesh is dragged. However, dragging leaves a trail of perturbed vertex positions and can produce small undulations in the surface when it is not perfectly flat. The detail left behind on a geometrically textured surface is not guaranteed to reflect the original texture at all. It also requires dragging with a small displacement and cannot support large displacements including movement to arbitrary target locations or other surfaces like a general cut and paste scheme can.

Hence, we consider the following as desirable properties for an approach to sliding relief features on surfaces:

- Supports various surface representations. **Representation-independence** is not only important for maintaining flexibility of the approach but it also affords a streamlined

implementation. Existing approaches solve the many smaller sub-problems for the many steps of the overall relief feature transfer process by borrowing solutions and techniques from many approaches each of which operate under their own set of limiting assumptions (such as the supported surface representations, kinds of surface shape, surface topology, etc.) or require special tweaking and tuning to work. For example, if a fairing approach that operates on subdivision surfaces were to use the result of a hole-filling approach that generates irregular triangle meshes, a conversion from triangle meshes to subdivision surfaces would be required.

- Preservation of the **connectivity** graph on the sliding path. Other modeling operations may be linked to the existing connectivity, e.g. a parameterization supporting textures; hence, we would like to preserve it if possible.

- Preservation of or control over treatment of surface details (**geometry**) on the sliding path. The result of a executing a feature sliding operation should be independent of the path taken if the initial and final positions are the same. Consider the case where the designer slides a feature away from and back to its initial position. Ideally, there should be no change in the model. Hence, we moreover propose that the result of a sliding operation should be equivalent to cutting and pasting from the original position directly to the final position as if the intermediate sliding was skipped.

- Ability to slide at **arbitrary displacements** across the surface. This essentially implies the ability to cut and paste between two arbitrary locations on a surface and even suggests the ability to cut and paste between different surfaces or models, provided that the slid feature fits on the target surface region in either case.

9.1.1 Our basic approach

To achieve the desirable properties we have set forth, we propose to view the problem of relief feature transfer using the following metaphor. We liken our basic approach to that of taking photographs with a camera and printing it out on some other medium. Because we are dealing with 3D surface geometry, a more fitting real-life metaphor for describing the overall approach is that of a 3D range scanner (e.g. a range finder camera) as the capture device and a 3D deposition device (e.g. a 3D printer) as the application device. Based on this metaphor, we can capture (or copy) geometric relief information by placing the virtual *scanner* at the source surface and performing a *scan*. Likewise, we can apply (or paste) the relief to the target surface by placing the virtual *depositor* and performing a *deposition*. Facilitating the information transfer between the scanner and the depositor is an intermediate representation of the relief which we call an *imprint* (Figure 9.1).

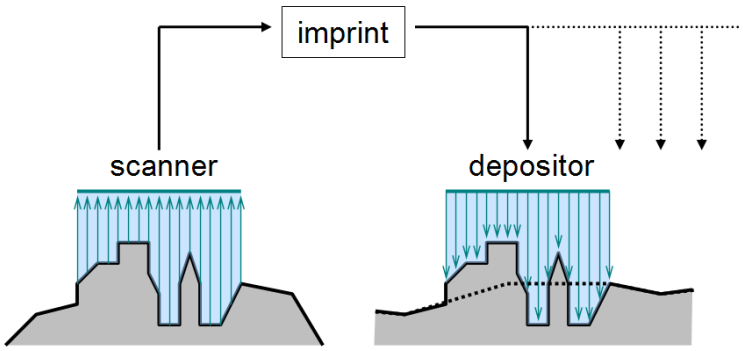

Figure 9.1: Imprint-based relief feature transfer.

To make a scan, we capture depth information from the source surface with respect to the scanner (defined by a pose and parameters). Based on the captured depth information,

we compute a base surface by fitting or filtering the original surface. Interpreting the original surface as a base surface with a relief feature on top, we can use this base surface to extract the relief feature information from the original surface by subtracting out the base surface. This relief information is the imprint.

To perform a deposition, we capture depth information from the target surface with respect to the depositor and we compute a base surface. Note that these steps are identical to the initial steps for making a scan. We then augment the computed base with the imprint information, resulting in a new depth field with the new feature applied. We use this depth field information in conjunction with the depositor (including its pose and parameters) to update the geometry of the surface.

Before we make a scan or perform a deposition, we need to place the scanner or the depositor device with respect to the model. To describe how to do this, let us take the metaphor further and let each device have several properties: imprint resolution, device size, device position, and device orientation. Imprint resolution indicates the number of samples, device size is analogous to the field of view and defines the spacing or spread of the samples, and position and orientation describe where the device is placed and in what direction it is pointed. We will provide more details and a precise definition of these properties later after we formulate the problem and define our imprint representation.

9.1.2 Contributions

With the scanner/depositor metaphor as a basis, we propose a relief feature sliding approach based on a dynamic regular discrete re-sampling of the surface which has the desired properties we have set forth. While the surface representation itself is not re-

sampled, the transfer of shape information is facilitated by an imprint, a regular discrete sampling of the surface. Specifically, we make the following contributions:

1 - We propose a new interpretation for 2D rectangular images, which we call ***imprint space***, which can be used as a representation to transfer geometric relief features from one surface to another.

2 - We present an approach which we call ***imprint-mapping*** that is based on imprint space to extract, process, and transfer relief features from one surface to another.

3 - Based on our imprint-mapping approach, we describe how to copy, paste, cut, delete, move, and slide relief features on surfaces.

9.1.3 Organization

The approach we propose has three main steps: (1) source/target specification, (2) extraction and processing, and (3) reconstruction. We address these three steps in the remainder of this chapter, organizing the discussion as follows. We first formulate the problem in terms of the prior art. We then define imprint space and our imprint representation for encoding relief features on surfaces. In defining the imprint representation, we show how to perform source/target specification in the context of our metaphor in order to extract geometric information into imprint space. We then show how to extract, encode, compose, and reconstruct relief features using imprints. Using these techniques as a basis, we describe how to copy, paste, cut, delete, move, and slide relief features on surfaces. In Chapter 10, we provide details on a practical implementation of our scheme and give results.

9.2 Problem formulation and background

We define our problem using a notation similar to [Biermann et al. 2002]. In defining the relief feature sliding problem, we assume that the surface containing the feature we wish to slide is homeomorphic to a disc and hence can be parameterized onto a planar domain $P \subset \Re^2$. Note that our approach will not require this parameterization to actually be computed. Nevertheless, we say that original surface **f**(x), its base surface **b**(x), and the resulting detail surface **d**(x) can be related by **d**(x) = **f**(x) – **b**(x), where x is a point in the planar domain. Hence, extracting a relief feature requires separating the original surface into detail and base components by (1) computing **b** as the base of **f**, (2) mapping points on **f** to points on **b** (i.e. each point **f**(x) is mapped to a point **b**(x)), and (3) computing **d** as the difference between **f** and **b** (i.e. **d**(x) = **f**(x) – **b**(x)). A relief feature can then be transferred by (1) locally recording $\mathbf{d}_1$ and replacing $\mathbf{f}_1$ with $\mathbf{b}_1$ at the source surface region (cut operation) and (2) replacing the detail part $\mathbf{d}_2$ of the target region with the detail part $\mathbf{d}_1$ of the source (paste operation).

We have pointed out that our goal of sliding a relief feature on a surface can be accomplished by successively invoking cut and paste operations on a local subset of the surface containing the feature. Barghiel et al. [1995] formulate this idea of local cut and paste more precisely using the formalism of displacement mapping. Here we borrow their description: True displacement mapping involves a vector-valued function **d**(r, s) defined over a compact domain $(r, s) \in \check{D}$, a 1-1 transformation T (e.g. a texture parameterization) of that domain into the domain $(u, v) \in \check{S}$ of a surface (point-valued) **b**(u, v), and the resulting composition **f**(u, v) = **b**(u, v) + **d**(T^{-1}(u, v)). Hence, the continuity of **f** depends on the continuity of **b**, T, and **d**. In the surface modeling setting we consider, the purpose

of employing a displacement map is to achieve the pasting of surface detail D(r, s) onto a base surface **b**(u, v). The function **d** is constructed from the difference between D and a reference surface. For convenience, the domain of D may be embedded into the range of D; that is, the points on D are identified with points in the space {(r, s, t)}, and the domain of D is identified with the plane (r, s, 0) by some convenient homeomorphism. This plane is often taken as the reference surface and hence **d**(r, s) = D(r, s) − (r, s, 0). The composition yielding **f** is then implemented with respect to a coordinate frame {(r, s, 0), **i**, **j**, **k**} appropriate to **d**, D and some manifold coordinate frame {**b**(u, v), **l**(u, v), **m**(u, v), **n**(u, v)} defined over **b** (with **n**(u, v) taken as the normal to **b**(u, v)). That is, $\alpha=\mathbf{i}\cdot\mathbf{d}(r, s)$, $\beta=\mathbf{j}\cdot\mathbf{d}(r, s)$, and $\gamma=\mathbf{k}\cdot\mathbf{d}(r, s)$ (which can be written as $\{\alpha, \beta, \gamma\}^T = \{i, j, k\}^T\cdot\mathbf{d}(r, s)$), which is to say $\mathbf{d}(r, s) = \alpha\mathbf{i} + \beta\mathbf{j} + \gamma\mathbf{k}$; then, $\mathbf{f}(u, v) = \mathbf{b}(u, v) + \alpha\mathbf{l}(u, v) + \beta\mathbf{m}(u, v) + \gamma\mathbf{n}(u, v)$. The mapping T should not only map (u, v) smoothly into (r, s) but it should also provide a smooth mapping from {(r, s, 0), **i**, **j**, **k**} to {**b**(u, v), **l**(u, v), **m**(u, v), **n**(u, v)}. The elements of this setting are shown in Figure 9.2, and except for the third dimension, this is exactly the setting for texture mapping [Rogers 1985].

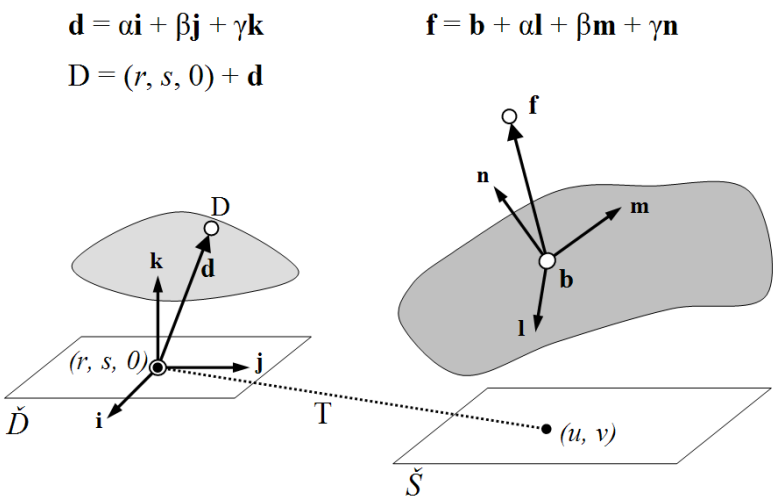

Figure 9.2: Displacement mapping. (Adapted from [Barghiel et al. 1995] Figure 1.)

9.3 Imprints

As a simplification of the general formulation of displacement mapping, we propose to view D as a height field or depth map taken with respect to the surface **f** in some direction N. This simplifies the formulation in several ways. First, instead of using a detail vector **d**(r, s) defined as the vector difference between D(r, s) and (r, s, 0) with respect to coordinate frame {(r, s, 0), **i**, **j**, **k**}, we use a scalar value d'(r, s) to encode surface detail information. We may say that d'(r, s) lies in *imprint space*, which we define as the space of all possible functions g(r, s) having an *imprint domain* $D_I \subset \check{D}$ and scalar range $g \in \Re$ defined for all $(r, s) \in D_I$. Likewise, all surface detail is defined with respect to a single direction N defined with respect to **f** (or **b**); hence, there is no need to compute the local coordinate frame {**b**(u, v), **l**(u, v), **m**(u, v), **n**(u, v)} and compute **f**(u, v) = **b**(u, v) + α**l**(u, v) + β**m**(u, v) + γ**n**(u, v) using the passed detail information (α, β, γ). Instead, points on **f** are expressed as **f**(u, v) = **b**(u, v) + d'(T(u, v))N. Then, we propose to define the mapping between $(r, s) \in D_I$ and $(u, v) \in S_I \subset \check{S}$ (where S_I is the domain of the region of

interest on **f**) using the parameters and placement of an *imprint device* (a depositor/scanner device in our metaphor).

An imprint device is defined by specifying an imprint resolution and a device size, position, and orientation. To define the details of these parameters, we first define an *imprint* as a set of scalars d'(r, s) corresponding to an imprint domain D_I. Note that as a practical representation for an imprint, we store samples from a discrete sampling of D_I and for simplicity use a sampling with a regular rectangular spacing having its resolution defined by width w and height h. This width and height are then the parameters which define the imprint resolution for an imprint device.

The device size, position, and orientation define how the imprint values sample the surface, that is, they define how the domain $D_I \subset \check{D}$ of the imprint maps to the domain $S_I \subset \check{S}$ of the surface. Specifically, size, position, and orientation can be described by a single frame $V = \{O_V, I_V, J_V, K_V\}$ which we call the *device frame*. The position O_V, the size $\{||I_V||, ||J_V||\}$, and the orientation $\{I_V, J_V, K_V\}$ can be used to define a set of sampling rays R(r, s), each ray having direction $N = K_V/||K_V||$ and a source point $P(r, s) = O_V + rI_V + sJ_V$. The surface f is then sampled as a function g(r, s) in imprint space (i.e. g is a scalar-valued function with domain $(r, s) \in D_I$) such that $\mathbf{f}(u, v) = \mathbf{f}(r, s) = P(r, s) + g(r, s)N$ for $(r, s) \in D_I$. Notice that the mapping between (u, v) and (r, s) is now (u=r, v=s). As a further simplification, we define normalized samples D(r, s) in imprint space as $D(r, s) = g(r, s)/||K_V||$ and hence $\mathbf{f}(u, v) = P(r, s) + D(r, s)K_V = O_V + rI_V + sJ_V + D(r, s)K_V$ (Figure 9.3).

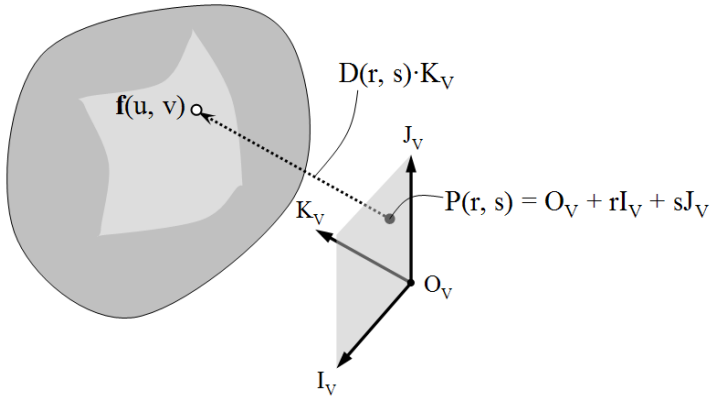

Figure 9.3: Imprint samping.

The fundamental difference between our imprint-mapping approach and prior approaches that are similar to displacement mapping (e.g. [Biermann et al. 2002], [Sorkine et al. 2004], and [Zatzarinni et al. 2009]) lies in the offset directions from the base surface which are used to encode the relief. Specifically, approaches similar to or based on displacement mapping use *intrinsic* surface information, e.g. the surface normal, while our imprint-based approach uses directions based on a single viewpoint and hence is *extrinsic* in this sense. An intrinsic approach can process relief features which wrap and follow a surface but the relief features may be subject to distortion and self-intersections because of convexities and concavities in the base surface shape (Figure 9.4). An extrinsic approach such as imprint-mapping can avoid such distortion and self-intersections since it restricts surface perturbation to be in a single direction (Figure 9.5a and Figure 9.5b). An extrinsic approach based on perspective projection (non-parallel rays) limits distortion to the degree of perspective and avoids self-intersections as long as the surface perturbation remains on one side of the view point (Figure 9.5c. The main limitation of a single-view extrinsic approach is that the surface region of interest

containing the relief feature must be front (same) facing from the viewpoint. This implies that overhangs with respect to the viewpoint are not supported (Figure 9.5d).

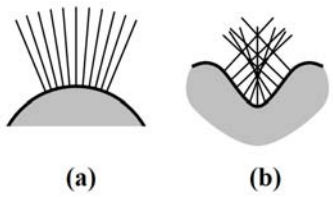

Figure 9.4: Intrinsic relief encodings such as an encoding based on surface normal directions may cause a relief to be distorted due to normal spreading (a) or have self-intersections (b).

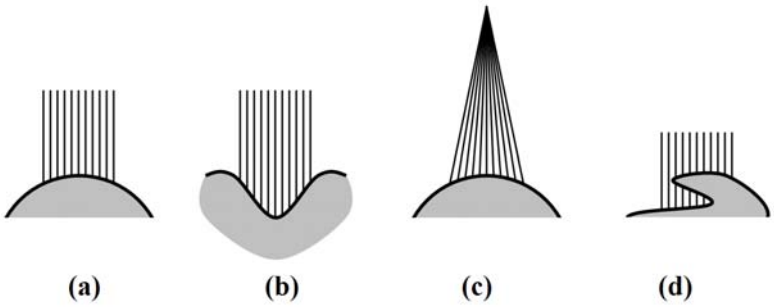

Figure 9.5: An extrinsic relief encoding can avoid self-intersections in the relief by construction (a & b). Encoding directions from a single viewpoint still avoids self-intersections on one side of the view point (c). Overhangs with respect to the view direction are not considered valid imprints from the given view point (d).

9.4 Source/target specification

Placing a scanner/depositor device can be achieved by specifying a device frame V. This defines a mapping between imprint space and the surface in model space which we can use to capture an imprint, that is, to sample the surface into imprint space. Given a device frame, we specify the source feature region by specifying a region in the imprint domain $D_I \subset \check{D}$. This can be achieved by defining a closed 2D curve $\{(r_0, s_0), (r_1, s_1), \ldots, (r_n=r_0, s_n=s_0)\}$ of n points $(r_i, s_i) \in \check{D}$. This corresponds with a closed 2D curve of n points

$\mathbf{f}(u_i, v_i)$ on the surface (Figure 9.6). Note that we do not actually need to compute the curve on the surface in order to scan and deposit imprints. The designer may specify this curve either on the surface in model space or in directly imprint space with respect to a device frame. For example, the designer can specify a device frame by manipulating a virtual camera, e.g. the camera used for rendering. The curve can then be specified by any number of methods, e.g. sketching a loop or drawing a series of points.

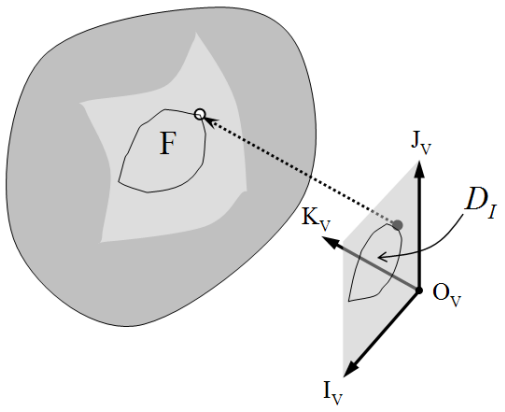

Figure 9.6: Given a device frame $V = \{O_V, I_V, J_V, K_V\}$, a curve in the parametric domain $\check{D}$ corresponds to a curve on the surface. The curve defines the imprint domain D_I which corresponds to a region F on the surface containing the feature of interest.

9.5 Imprint based relief processing

We now show how to extract, encode, compose, and reconstruct relief features using imprints. The basic displacement mapping formalism is used by Barghiel et al. [1995] to describe surface pasting. We propose to use our simplified displacement mapping formalism to not only paste reliefs but to extract them. Our *imprint-mapping* approach essentially embeds the imprint domain in model space; hence, to perform an operation on the imprint in imprint space is essentially to perform an operation on a surface in model

space. Imprint based relief processing has the following basic steps (Figure 9.7): (1) A surface is scanned into imprint space by positioning a scanner device (*source specification*) and sampling the surface (*scanning*). (2) The imprint is processed by computing a base surface in imprint space (*base computation*), encoding the feature as offsets with respect to the base (*feature encoding*), and editing and composing the encoded offset imprint possibly mixing with other encoded offset imprints (*composition*). The key point is that all of this processing is done in imprint space. (3) The relief is deposited on the target surface by positioning a depositor device (*target specification*) and producing point samples of the new surface in model space for each imprint sample. This involves first decoding the offset imprint back to a depth map imprint (*feature decoding*) and then using the device parameters to generate a set of samples in model space. A representation-dependent warping or fitting approach can then use these samples to reconstruct the edited surface (*reconstruction*). We now describe each of these basic steps in more detail.

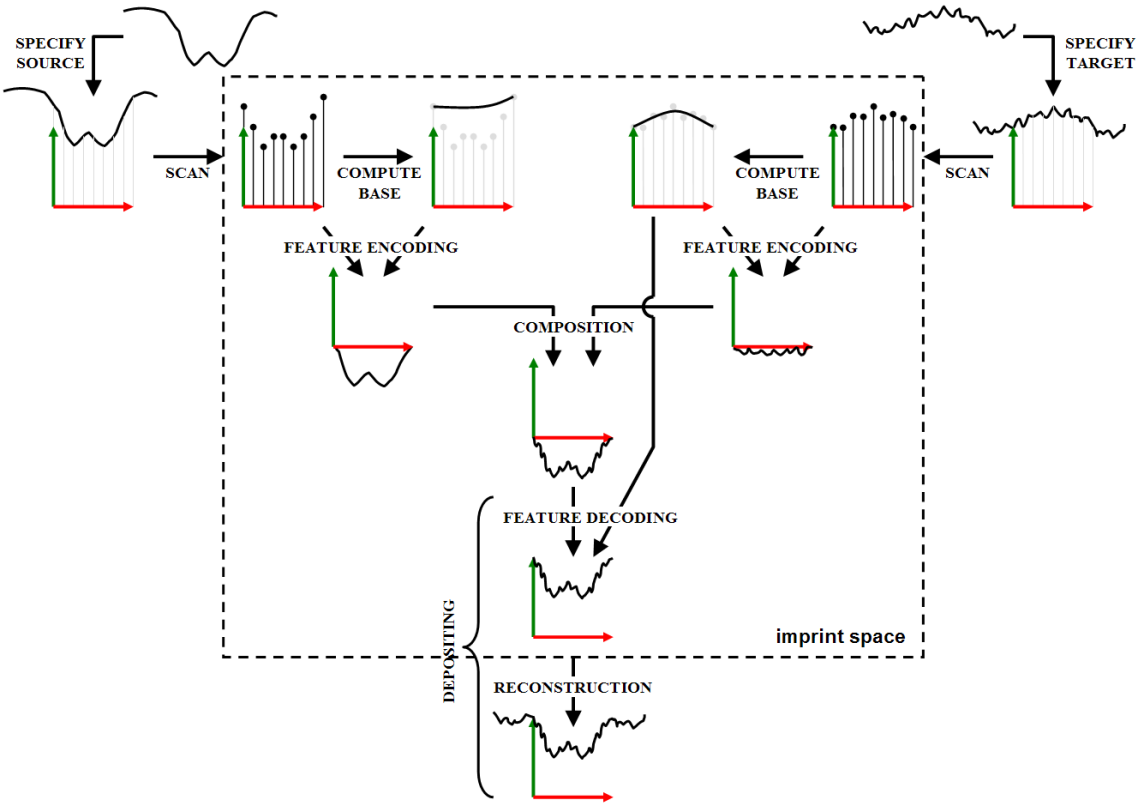

Figure 9.7: Imprint-based relief processing.

9.5.1 Scanning

To perform a scan, depth values are sampled from the surface given a device frame V. The resulting set of depth samples D(r, s) for $(r, s) \in D_I$ is the scanned imprint. An imprint, essentially being a depth image, can be created with approaches similar to rendering approaches. In particular, we consider the two basic rendering approaches for image generation: ray tracing and rasterization.

A ray tracing approach casts rays to sample the scene and determine pixel values of an image. Similarly, we can determine depth values of an imprint by casting rays and intersecting them with the surface to be sampled. Given a device frame V, the surface is sampled as $D(r, s) = (\mathbf{f}(r, s) - P(r, s)) \bullet K_V$. This can be achieved by performing a ray-

surface intersection computation D(r, s) = intersect(**f**, R(r, s)), where R(r, s) is the ray {P(r, s), K_V} and the function intersect() returns the parameter t such that Q = P + tK_V, where Q is the first intersection of ray (P, K_V) with surface **f**.

Conversely, a rasterization approach back projects points in the scene to a 2D image plane and often the rest of the pixels of the image are obtained via a form of scan conversion and interpolation. Similarly, points in the surface model can contribute depth samples to the imprint and additional imprint samples can be obtained using interpolation. A point Q_i on the surface in model space contributes a value $D(r, s) = (K_V \bullet O_V Q_i)/||K_V||^2$, where $r = (I_V \bullet O_V Q_i)/||I_V||^2$ and $s = (J_V \bullet O_V Qi)/||J_V||^2$. (In general, $||A||^2$ can be computed as A•A to avoid computing a square root when finding the length ||A||.) Note that these expressions for r and s show how a device frame V gives us a parameterization of points Q_i on a surface to the parametric domain D_I. We essentially perform a change of coordinate systems from the model space to the parametric imprint space.

The choice of which scanning approach is more suitable depends on the surface model being sampled. The number of points defining the surface model determines the number of points a rasterization approach needs to project. Hence, the amount of computation is directly proportional to the number of points projected. A ray casting approach would for each ray consider all the objects in the model or simply the surface element of interest. While this is more expensive than basic rasterization, it allows samples to be taken from more than one object in the scene or assembly model. For example, the valley crease between a triangle mesh and a parametric sphere could be sampled as if it was represented by a single surface. Similarly, a rasterization approach based on the idea of z-buffering can also provide a representation-independent way of

sampling the surface. A scene or assembly can be rasterized into an imprint simply by rendering it and extracting the content of the depth buffer from the GPU. This makes the imprint approach flexible in the sense that irrespective of the representation, any model component that can be rendered into the depth buffer can affect the shape of the scanned imprint.

9.5.2 Base computation

Since the surface is represented as a height field on a regular grid, a number of approaches are applicable for computing a base surface, either as a reference surface, a replacement surface, or both. For example, plane fitting (e.g. least squares plane fitting), smoothing (e.g. using a Gaussian filter), and interpolation-based hole filling (e.g. [Farbman et al. 2009]) are some of the possibilities. Many base computation approaches based on fairing, fitting, and interpolation have been proposed which make use of the regular grid structure; hence, we refer the reader to the prior art (see the literature review in Section 8.2.3). In our formalism, we may summarize any number of possible base finding approaches as a function B=base(S) which computes a base surface imprint B for imprint S.

The base surface computed by the function base() directly determines the shape of the relief in terms of both what is extracted and what is deposited. For example, a planar base surface at both extraction and deposition would result in preservation of feature shape (rigid transfer), while a base that follows the lower frequency shape of surface would allow details to be “peeled” off of a surface.

The choice of base surface also determines the discrepancy at the feature boundary. For example, a plane fit base surface on a curved surface generally results in

discontinuities at the boundary, while a smoothing approach could enforce continuity at the boundary using by constraining boundary values. Independent of the approach used for base computation, we provide additional means for obtaining boundary continuity by composition of the source relief feature with the target relief feature via a blending function (as discussed in Section 9.5.4).

The appropriate choice of base surface depends on the application and the needs of the designer. Hence, in the framework of our approach, we allow the base surface computation to be customizable according to the needs of the designer.

9.5.3 Relief feature encoding

Given a base surface imprint, we can encode the relief by subtracting the base imprint B from the surface imprint D resulting in offset imprint I = D – B. This is a per-sample subtraction, i.e. I(r, s) = D(r, s) – B(r, s).

9.5.4 Composition

Given an imprint I, we may edit it as an image by modifying the values of the samples I(r, s). For pasting reliefs, we may wish to mix surface detail from the source feature with surface detail at the target region to, for example, achieve a smooth blend at or near the feature boundary. This can be achieved by scanning imprints at both the source and target surface regions, encoding the offsets (difference between the depth scans and the respective computed base surfaces) as imprints (I_S and I_T), and using a blending function t(r, s). Blending is then expressed as a weighted average: $I_C(r, s) = t(r, s)I_S(r, s) + (1 – t(r, s))I_T(r, s)$. Note that we may also compose an arbitrary number of imprints by averaging or using some other statistical analysis. This idea can be used to

perform homogenization of a set of features, i.e. feature regularization of a pattern of relief features on a surface.

9.5.5 Depositing

Given an offset imprint, e.g. composed imprint I_C, the surface can be decoded as $F_C = B_T + I_C$, i.e. $F_C(r, s) = B_T(r, s) + I_C(r, s)$, where $B_T = \text{base}(F_T)$. Now given a device frame $V = \{O_V, I_V, J_V, K_V\}$, we can compute samples in model space: $\mathbf{f}_C(r, s) = O_V + rI_V + sJ_V + F_C(r, s)K_V$. These samples $\mathbf{f}_C$ can then be used by a warping or fitting approach (the particular choice of which is dependent upon the surface representation) to deposit or paste the relief feature at the target surface.

Approaches have been proposed for various representations which perturb the surface geometry to conform to some shape, e.g. displaced subdivision surfaces [Lee et al. 2000] and Pointshop 3D [Zwicker et al. 2002]. Furthermore, numerous approaches have been proposed for surface reconstruction from point clouds [Hoppe et al. 1992; Ilic and Fua 2006]. Again, we point out that the actual modification of the surface geometry is dependent upon the surface representation. The key point is that we can generate sample points in model space from an imprint given a device frame.

9.6 Relief editing operations

Based on our basic imprint approach, the following operations for relief features are available: copy, paste, cut, delete, move, and slide. We define a small set of operations to help us describe how each are implemented. The operation scan(V, **f**) returns a depth map given device frame V and surface **f**. The operation base(D) returns a base surface imprint given imprint D. The operation compose(I_1, t, I_2) mixes I_1 and I_2 using weights t. The

operation deposit(V, **f**, D) warps the surface **f** given device frame V and depth map imprint D. We assume that V_i is a device frame, $\mathbf{f}_i$ is a surface, and D_i and I_i are imprints. Note that while we interpret D_i as a displacement map and I_i as an offset map, both are imprints and have the same representation. Given this notation, we can describe the following set of unary editing operations for relief features; that is, each operates on a single feature:

Copy: The copy operation (Figure 9.8) obtains an imprint I_S (representing the relief feature as an offset from the base) from surface F_S given device frame V_S.

I_S = copy(V_S, $\mathbf{f}_S$) { D = scan(V_S, $\mathbf{f}_S$); I_S = D – base(D); }

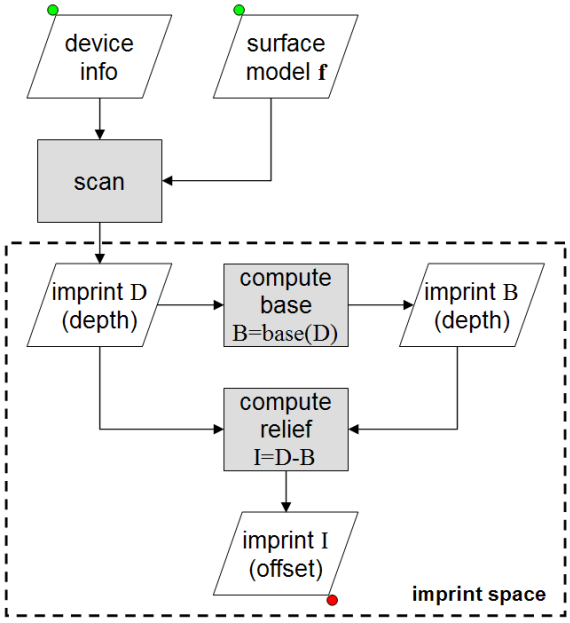

Figure 9.8: Algorithm for imprint-based relief copy operation.

Delete: The delete operation (Figure 9.9) removes the relief indicated by device frame V_S on surface $\mathbf{f}_S$ by replacing the region of interest with the base surface.

delete(V_S, $\mathbf{f}_S$) { D = scan(V_S, $\mathbf{f}_S$); deposit(V_S, $\mathbf{f}_S$, base(D)); }

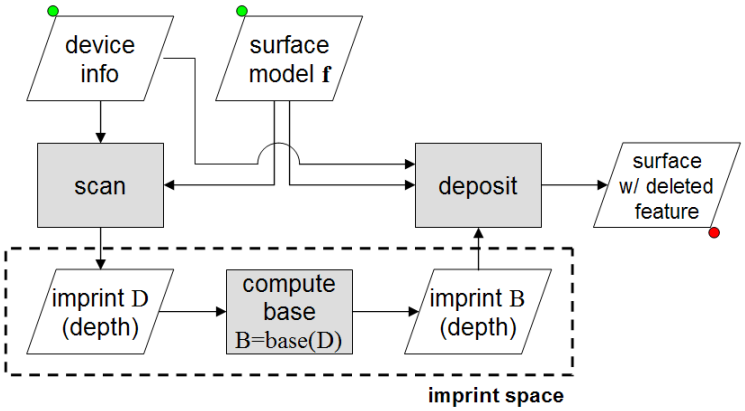

Figure 9.9: Algorithm for imprint-based relief delete operation.

Cut: The cut operation (Figure 9.10) is essentially a copy operation followed by a delete operation.

$I_S = \text{cut}(V_S, \mathbf{f}_S)$ { $D = \text{scan}(V_S, \mathbf{f}_S)$; $D_B = \text{base}(D)$; $I_S = D - D_B$; $\text{deposit}(V_S, \mathbf{f}_S, D_B)$; }

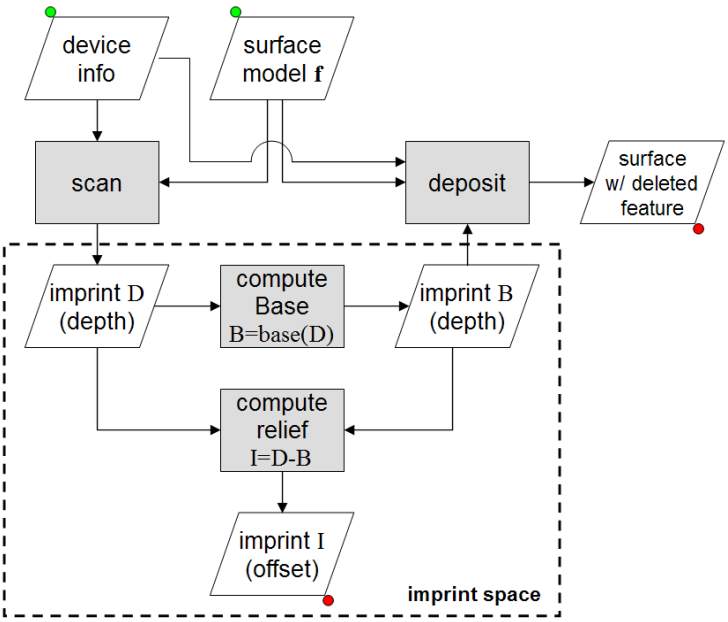

Figure 9.10: Algorithm for imprint-based relief cut operation.

Paste: The paste operation (Figure 9.11) essentially copies the relief at the target, mixes it with the relief to be pasted, and augments the target base with this composed relief.

paste(V_T, $\mathbf{f}_T$, I_S, t) { D = scan(V_T, $\mathbf{f}_T$); D_B = base(D); I_T = D – D_B; deposit(V_T, $\mathbf{f}_T$, D_B + compose(I_S, t, I_T)); }

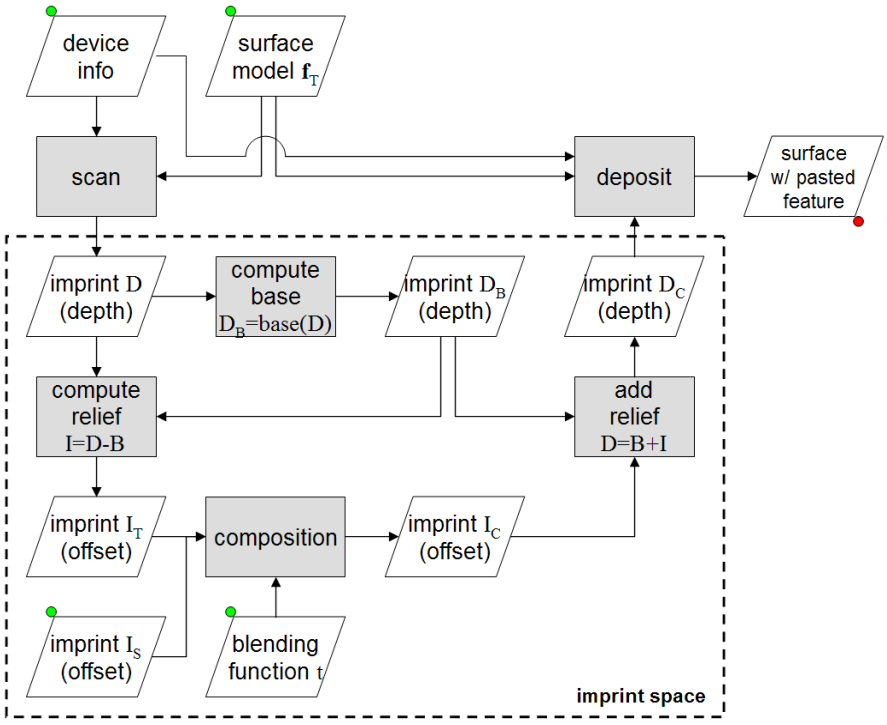

Figure 9.11: Algorithm for imprint-based relief paste operation.

Move: A move operation (Figure 9.12) is a cut followed by a paste.

move(V_S, $\mathbf{f}_S$, V_T, $\mathbf{f}_T$, t) { I_S = cut(V_S, $\mathbf{f}_S$); paste(V_T, $\mathbf{f}_T$, I_S, t); }

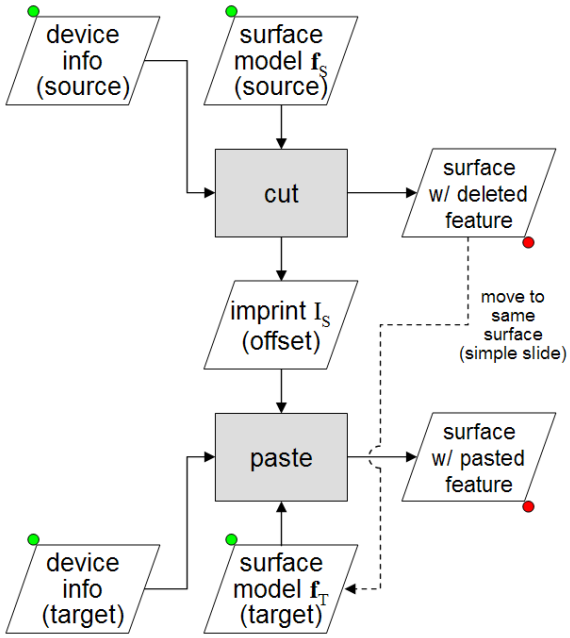

Figure 9.12: Algorithm for imprint-based relief move operation.

Slide: The slide operation (Figure 9.13) can be implemented as a series of one or more small moves. However, this simple method will cause the part of the surface on the slide path to be changed even after the feature has moved past it. Hence, for the purpose of sliding, we use a modified version of move. The idea is to copy the target surface before pasting and use this saved imprint as the replacement base when next moving the feature. Furthermore, only the initial feature is cut. Subsequent pasting uses the same original feature. We describe the sliding of a feature along path V_0, V_1, …, V_n (where $V_0 = V_S$ and $V_n = V_T$) on surface **f** as follows.

The initial move for sliding involves cutting the relief feature that we will slide to obtain I_S and pasting in the first new location defined by V_1. Note that we use a special paste operation paste2() that saves the depth map at the target location before pasting.

I_S = cut(V_0, **f**);
D_R = paste2(V_1, **f**, I_S, t);

Subsequent moves for sliding the feature to locations defined by V_i for i=2…n are achieved by restoring the surface to its pre-pasting state and again applying paste2() in the new location defined by V_i.

deposit(V_{i-1}, **f**, D_R);
D_R = paste2(V_i, **f**, I_S, t);

Note that paste2() is identical to paste() except that it returns the original depth map D. We define paste2() as:

D = paste2(V_T, $\mathbf{f}_T$, I_S, t) { D = scan(V_T, $\mathbf{f}_T$); D_B = base(D); I_T = D – D_B; deposit(V_T, $\mathbf{f}_T$, D_B + compose(I_S, t, I_T)); }

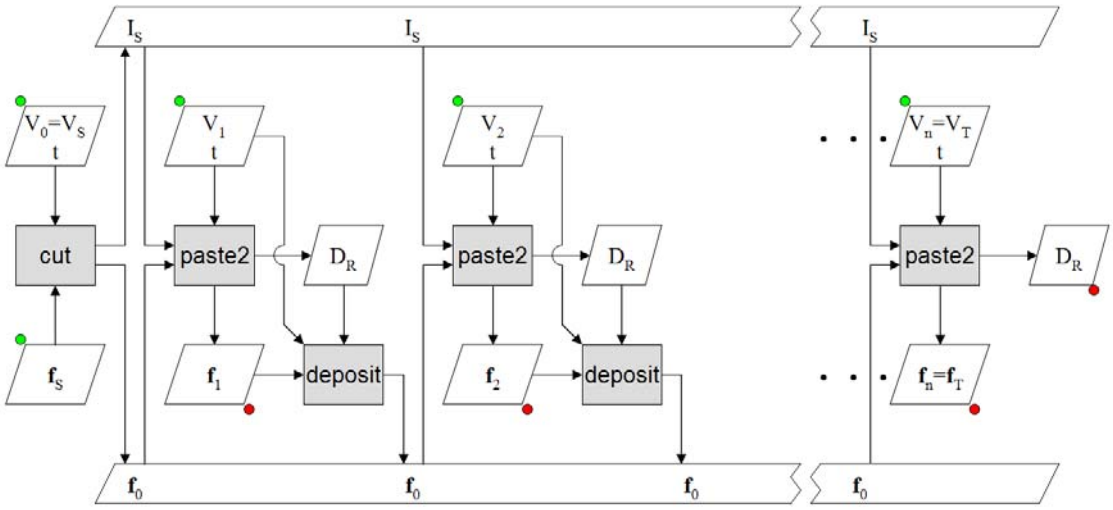

Figure 9.13: Algorithm for imprint-based relief slide operation.

9.6.1 Operations for multiple reliefs

In addition to these six unary operations which operate on single relief features, we may also define a set of operations which operate on multiple relief features, in particular, on regular patterns of relief features. Let $\boldsymbol{V} = \{V_1, V_2, \ldots, V_n\}$ be a set of n device frames and $\boldsymbol{I} = \{I_1, I_2, \ldots, I_n\}$ or $\boldsymbol{D} = \{D_1, D_2, \ldots, D_n\}$ be a set of n imprints.

Given a set of device frames $\boldsymbol{V}$, the operation $\boldsymbol{D}$ = scan($\boldsymbol{V}$, **f**) returns a set of depth map imprints $\boldsymbol{D}$ corresponding to device frames $\boldsymbol{V}$ and surface **f**. This n-ary scan operation is equivalent to applying the unary scan operation D_i = scan(V_i, **f**) for $1 \leq i \leq n$.

Likewise, deposit($\boldsymbol{V}$, **f**, $\boldsymbol{D}$) is equivalent to applying the unary deposit operation deposit(V_i, **f**, D_i) for $1 \leq i \leq n$.

In general, all of the unary operations have similarly corresponding n-ary operations, e.g. $\boldsymbol{I}$ = copy($\boldsymbol{V}$, **f**), delete(**V**, $\mathbf{f}_S$), etc.

Recognition: In order to edit a parametric pattern, the system must first recognize a sequence of feature instances. Recognition in general is a challenging problem and much work has been presented which addresses the problem in the context of reverse engineering and shape analysis [Várady et al. 1990], [Vandenbrande and Requicha 1993], [Regli et al. 1994], [Reed et al. 1995], [Traband et al. 1996], [Várady et al. 1997], [Lee and Kim 1998], [Thompson et al. 1999], [Benko et al. 2001], [Langbein et al. 2001b], [Mills et al. 2001], [O'Mara 2002], [Gao et al. 2003], [Langbein 2003], [Liu et al. 2006], [Liu et al. 2007b], [Pauly et al. 2008]. For recognizing regular patterns of relief features on surfaces, we propose a simple user-guided approach. In particular, given a set of device frames (i.e. a set of user guesses), we provide a snapping operation for registration which adjusts the device frames such that the set of relief features which they identify (i.e. the features which they are positioned to scan) are similar, that is their discrepancy is minimized. Given an initial set of device frames $\boldsymbol{V}$, the operation snap($\boldsymbol{V}$, **f**) adjusts the existing device frames in $\boldsymbol{V}$ such that disparity(I_i, I_j) is minimized, given that $\boldsymbol{I}$ = scan($\boldsymbol{V}$, **f**), for $1 \leq i \neq j \leq n$. The amount by which V_i can be adjusted should be bounded since we

assume that the given frame "guesses" are approximately close to the desired frames. For instance, fix the scale and allow for rotation variation α and translation variation δ.

Such a snapping approach can be used to assist the designer in specifying individual frames to identify individual features. For example, in the modeling scenario in Figure 1.17, the designer specifies three frames A, B, and C in order to define two pattern transforms taken as the relative transform between A and B and between A and C, respectively. Once the designer has specified A, snapping can be used to help the designer precisely specify B and C.

The operation disparity(I_1, I_2) returns a quantitative value for the disparity/discrepency between imprints I_1 and I_2. One definition for disparity is the per-sample sum of square differences between I_1 and I_2, but other definitions for disparity are possible. Since the disparity function is used to assist the snapping, a good disparity function would penalize larger discrepancies greater than small ones. We expect that image registration algorithms [Zitová and Flusser 2003] could be employed to assist feature alignment and we leave further exploration to future work.

Feature regularization (beautification/homogenization): Feature regularization can be achieved by scanning all of the instances into imprints, using these imprints to compute a pattern leader (which is the unique prototype which represents all of the other instances in a regular pattern), and depositing the result.

There are various ways of computing a pattern leader. (1) The first and simplest approach is to simply pick one of the instances as the leader. This can be arbitrary, e.g. use the first instance by convention, or manually selected. (2) A second approach is to use a pre-defined model of the feature as the prototype. This model can be determined by

searching a shape database or selected from a palette of features by the user. (3) The third approach is to compute the pattern leader by computing the average of all of the instances. This approach directly incorporates the shape information present in the given feature instances. Using different averaging functions gives us the flexibility to use different statistical operations such as per-sample averaging, median, mode, min, max, etc. Since they are per-sample (per-pixel) operations, these operations do not average the actual shapes, but only their height maps. Note that these per-sample averaging approaches rely on good alignment among all the features being averaged to work well.

Frame regularization: Frame regularization can be achieved by cutting the reliefs, adjusting the frames, and re-pasting the reliefs. In the prior art chapter (Section 4.1), we have discussed several alternatives to computing steady sequences of frames, i.e. a set of frames $\boldsymbol{V}$ such that $V_i = AV_{i-1}$, where A and V are affinities represented by a matrix in homogeneous coordinates. An alternative approach is to defining regularly spaced frames with respect to the surface. Such an approach could take advantage of a surface parameterization or simply project a steady set of frames onto the surface. We leave the solution of this problem as future work.

Re-spacing: Adjusting the pattern transform re-spaces the frames in $\boldsymbol{V}$. The procedure for updating the relief feature instances is similar to the procedure for frame regularization: the reliefs are cut, the frames adjusted, and then the reliefs are re-pasted. As already mentioned, the main difference in the case of patterns of relief features is that the frames may be defined with respect to the surface. However, it is possible to define a set of camera frames "free floating" in space as determined by the pattern transform (the approach we take in our examples, e.g. Figure 1.17, Figure 10.17, and Figure 10.18). We

leave a full exploration of the problem of defining a set of frames regularly spaced on a surface as future work.

Pattern leader: Simultaneous editing of all of the instances is a matter of editing one imprint, deleting all of the reliefs, and pasting the one result for all the device frames in the pattern.

Exceptions: An exception can be invoked and maintained using the OCTOR approach presented in PART I of this thesis. The main additional challenge is how to pick instances of relief features on surfaces. A trivial approach is to use a graphical/iconic visual representation for the device frames; the designer could then click directly on the iconic widgets to pick an instance. Alternatively, the faces in the ROI of each relief feature instance could be associated with the respective device frame pointed at it. Then, a designer click on the face would be indexed to the appropriate device frame and thus pick the instance.

10 IMPLEMENTATION AND RESULTS

In this chapter we describe our prototype implementation of imprint based relief transfer and give results. We assume that the surface is represented as a triangle mesh. We represent an imprint as a rectangular image of w×h pixels, where each pixel is a real number. Our implementation employs the standard OpenGL rendering pipeline and our examples are rendered using perspective projection with a 90 degree field of view.

We can give an overview and a basic description of our implementation by following the imprint-based relief processing steps described in Section 9.5. First, we position the view frustum to view the source or target region and specify some region of the visible surface to be the region of interest (ROI) which contains the relief feature. We can then scan the ROI into a depth map imprint D_S by extracting the content of the depth buffer. Given D_S, we compute a base surface B_S=base(D_S) and obtain an offset imprint I_S=D_S-B_S. We can then edit I_S to change the shape of the relief feature. In particular, we can blend (compose) it with an offset imprint I_T obtained from the target region. The composed imprint I_C is then applied to the target base depth map imprint B_T to obtain depth map imprint D_C=B_T+I_C. Finally, each vertex in the ROI queries D_C to obtain a depth for vertical (in the direction of the view) adjustment. These steps are the building blocks of the imprint-based relief feature editing operations described in Section 9.6; hence, given an implementation of each component, we can use them to copy, delete, cut, paste, move, and slide relief features on surfaces.

We now discuss several steps in more detail: (1) *source/target specification* – how we specify the source or target surface region containing the relief feature of interest, (2)

scanning – how we sample the source surface into an imprint, (3) *base computation* – how we compute a base in imprint space, (4) *composition* – how we edit and combine imprints, and (5) *depositing* – how we modify the target surface shape according to an imprint.

10.1 Source/target specification

We specify the source or target region by placing the rendering camera in OpenGL (e.g. using gluLookAt()) and specifying a 2D boundary in screen space. For the designer, placing the camera means navigating the scene or manipulating the view of the model. Specifying a 2D boundary in screen space can be achieved by a standard 2D region selection method such as dragging a selection box, drawing a closed loop using the mouse, or simply clicking an ordered sequence of points on the screen which define a loop. Given the loop, we can define a containment mask of pixels with value 1 for pixel included in the ROI (i.e. in the interior of the loop) and with value 0 for other pixels. In our implementation, we use a selection box and hence the w×h imprint can be used to represents the ROI without the need of additional information to identify the actual imprint domain corresponding to the ROI. Note that the base computation step may require *boundary samples*, that is, samples just outside the ROI boundary, in order to maintain continuity at the boundary. A *boundary thickness* of b samples means that the actual dimensions of the imprint are $w_b \times h_b$, where $w_b=(w+2b)$ and $h_b=(h+2b)$. Henceforth in our discussion, when we use the dimensions $w_b \times h_b$, we are referring to a w×h imprint along with a boundary that is b samples thick. In Section 10.3 we describe how to compute boundary pixels for other ROI shapes.

A case may arise where the designer would like to edit a patch which has holes. Several scenarios are possible: (1) the surface model has holes, (2) there is a non-local obstruction in the view (an occluder which is not connected to the patch of interest), and (3) there is a local self-occlusion (within the patch there are overhangs with respect to the view and hence more than one point on the surface project to the same point in screen/imprint space). In the case of a non-local occluder, the system may simply hide (not render) portions of the surface which are not adjacent to the ROI patch. In the other two cases, we propose two possible solutions. First, the designer may simply select them and they can be removed from the ROI containment mask. However, this approach would not be ideal in the case where local self-occlusions may be difficult to visually discern. The designer may not even be aware of such surface characteristics. One way of automatically detecting these is to check the depth values for outliers, that is, to look for depth values which are statistically anomalies with respect to the rest of the field [Barnett and Lewis 1994]. For the case of thru-holes with no other geometry behind or in the case where we are interested in a region close to the silhouette edge of the model, we can explicitly look for imprint values which are equal to the default z-buffer value. Once such imprint pixels have been identified, the corresponding pixel in the containment mask is set to zero.

Figure 10.1 shows two examples of source/target specification in screen space using a 104×104 selection box to obtain a 100×100 imprint with a boundary thickness b=2.

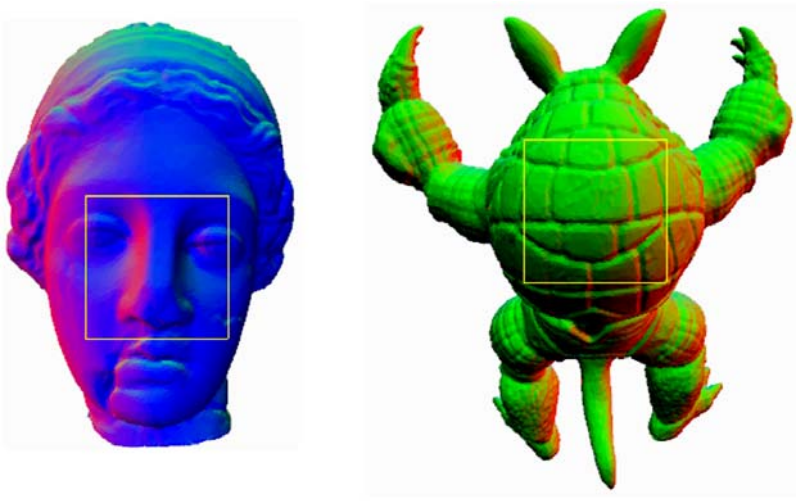

Figure 10.1: Source/target selection of (a) Igea head and (b) armadillo models is achieved using a screen space selection region.

10.2 Scanning

We scan a scene or assembly into an imprint by rendering it and extracting the content of the z-buffer (depth buffer) from the OpenGL graphics pipeline. We take the imprint D_S from a $w_b \times h_b$ subset of the z-buffer containing the ROI.

In our implementation we use perspective projection. For perspective projection in OpenGL, model space values Z_i in the range $[Z_n, Z_f]$ (where Z_n and Z_f are the near and far clipping plane distances, respectively) are mapped to z-buffer values z_i in the range [0, 1] using $z_i = (1/Z_n - 1/Z_i) / (1/Z_n - 1/Z_f)$. Due to this non-linear mapping, the z-buffer values give a distorted representation of the geometry. Hence, before performing any geometry processing in imprint space, we would like to convert the depth values back into a linear space, i.e. model space. To do this, we reverse the mapping using $Z_i = Z_n * Z_f / (Z_f - z_i * (Z_f - Z_n))$. Then, $D_S(r, s) = Z_i$, where $i = s * w_b + r$. Note that the purpose of this correction is not

to reverse the perspective projection of the z value, but to reverse the additional scalar mapping that OpenGL performs to attain better numerical fidelity in the depth buffer.

The imprints obtained by extracting the z-buffer region corresponding to the selections made in Figure 10.1a and Figure 10.1b and mapping the z-buffer values into model space depths are illustrated in Figure 10.2a and Figure 10.2d, respectively.

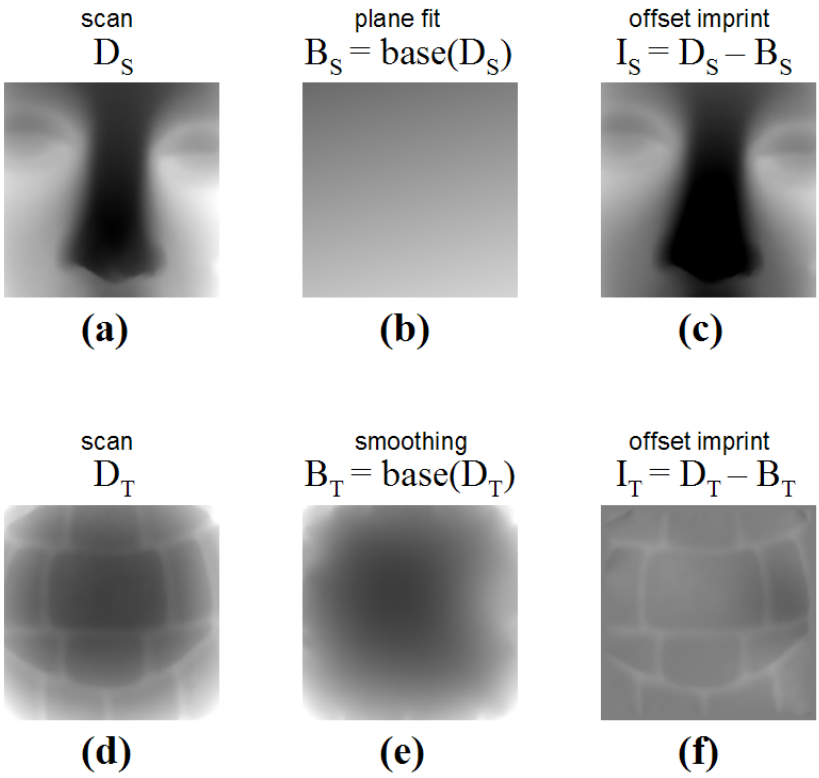

Figure 10.2: Imprints corresponding to the Igea head (a, b, c) and the armadillo model (d, e, f). Corresponding to the selections in Figure 10.1a and Figure 10.1b respectively, (a) and (d) are depth scans extracted from the z-buffer converted to model space depth values. (b) is the base surface imprint computed from (a) using plane fiting of border samples and (c) is the computed source offset imprint. (e) is the base surface imprint computed from (d) using smoothing and (f) is the computed target offset imprint. Note that the source or target roles are ambiguous until the composition and depositing steps.

10.3 Base computation

Given an imprint represented as a rectangular $w_b \times h_b$ set D of depth samples of a surface **f**, we wish to compute a base surface to **f** as a $w_b \times h_b$ set B of depth samples corresponding to D. Specifically, we define the boundary thickness as 2 samples/pixels (to facilitate continuity at the boundary) and hence $w_b=w+4$ and $h_b=h+4$. The goal is to compute the base surface imprint B=base(D) using the boundary of D as constraints. Once we have computed B, we can obtain offset imprint I representing the relief feature using per-pixel subtraction: $I(r, s) = D(r, s) - B(r, s)$ for $0 \leq r \leq w$ and $0 \leq s \leq h$ (e.g. Figure 10.2c and Figure 10.2f).

We have implemented the following base computation techniques: (1) plane fitting and (2) smoothing the imprint. Note that for our proof-of-concept prototype we use a rectangular ROI. For a ROI shape represented by a closed curve defined by a user provided lasso, we could define a containment mask from the lasso. That is, pixels in the interior of the lasso get a value of 1 and other pixels get a value of 0. Then, we have several options for determining which samples are the boundary samples. In the case of smoothing, the boundary samples needed could be computed by using a morphological grow operation to grow the region (defined by 1's in the containment mask) using the shape of the smoothing mask as the growing kernel and then subtracting out the original ROI pixels. In the case of finding boundary samples for plane-fitting, we can simply taking pixels touching pixels (in the up, down, left, right, and four diagonal directions) in the ROI (a one-ring neighborhood) or also add pixels touching the one-ring neighborhood (i.e. two-ring neighborhood of pixels). Overall, the ROI shape determines which pixels

are processed in the imprint processing steps and which vertices are processed in the depositing step.

10.3.1 Plane fitting

Instead of fitting a plane to all of the samples of an imprint, we fit a plane to just the border samples. An imprint for the reference base can then be obtained from the computed plane via sampling. While a planar base may not be ideal for curved surfaces in general, we implement plane-fitting to show that the approach is flexible enough to allow for different choices of base surface. If the designer is working with flat surfaces, the designer may wish to have such an option for better results as in Figure 1.17.

To fit a plane to a set of samples, we find a plane which minimizes the sum of square distances of the plane to the set of sample points. The least-squares plane fitting recipe we use is as follows [MathForum]. Given a set of n height pixels $P_i = (u_i, v_i)$, we construct a set of n points $W_i = (x_i, y_i, z_i)$, where $x_i = u_i$, $y_i = v_i$, and $z_i = D(u_i, v_i)$. We find the center point C_P of the plane as the centroid (average) of all of the points $C_P = \frac{1}{n}\sum_{i=1}^{n} W_i$.

We then build n×3 matrix M as:

$$M = \begin{bmatrix} x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ x_2 - x_0 & y_2 - y_0 & z_2 - z_0 \\ x_3 - x_0 & y_3 - y_0 & z_3 - z_0 \\ \vdots & \vdots & \vdots \\ x_n - x_0 & y_n - y_0 & z_n - z_0 \end{bmatrix},$$

where $(x_0, y_0, z_0) = C_P$ and $(x_i, y_i, z_i) \in W$, for $1 \leq i \leq n$. We perform singular value decomposition (SVD) on matrix $M = USV^T$ using JAMA (Java Matrix libraries) [Jama].

Then the normal vector N_P of the plane is the vector from V corresponding to the smallest singular value (on the diagonal of S).

Given point on the plane C_P and vector N_P normal to the plane and an (x, y) pair of coordinates, we would like to compute the z value of the plane. From the plane equation Ax+By+Cz+D=0, we obtain an expression for z given x and y: z=-(D/C)-(A/C)*x-(B/C)*y. We compute $D=-C_P \bullet N_P$ and assign (A, B, C) = N_P, that is, $A=N_P.x$, $B=N_P.y$, and $C=N_P.z$.

Figure 10.2b shows the result of fitting a plane to the border pixel depth samples of the scanned depth imprint in Figure 10.2a and then sampling the plane back into an imprint.

10.3.2 Imprint smoothing

For computing a base surface imprint B using smoothing, we smooth the imprint while keeping the boundary fixed to enforce certain continuity constraints at the boundary. The type of continuity produced at the boundary depends on the size and type of smoothing kernel used. Specifically, we wish to solve the following problem. We are given a height field on a regular rectangular grid which includes two rows of constraints on the boundary. These two rows are used to define constraints for not only the positions of the computed base but also the slope of the base surface at the boundary. We wish to filter the height field as such that it is smooth while respecting the boundary constraints.

In our implementation, we run an iterative process where during each iteration, we move each pixel sample value towards a weighted average of some of its neighbor pixels. Specifically, we compute the cubic fit of its four vertical neighbors (two above and two below) and its four horizontal neighbors (two to the left and two to the right). Then we

use a conjugate gradient inspired solver where the stopping condition is that the discrepancy between each sample and its ideal cubic fit value is sufficiently close to this discrepancy computed at its neighbors.

Figure 10.2e shows the result of running 1000 iterations of our cubic fit smoothing process on the scanned imprint in Figure 10.2d while fixing two rows of border pixel samples constant.

Note that we have implemented a smoothing scheme for the purposes of proof-of-concept. The iterative sample tweaking approach we have implemented is not necessarily optimal in terms of processing time and convergence. Others have solved such problems by defining an adjacency matrix of weights or a similar system of equations including constraints and solving the matrix system. We refer the reader to [Schneider and Kobbelt 2001], [Sorkine et al. 2004], and [Desbrun et al. 1999] for a discussion of alternative weighting schemes and solvers for the optimization.

10.4 Composition

Relief features can be edited or combined in imprint space. We represent an imprint using a w×h image of real numbers; hence, imprint editing is similar to image editing. Hence, we can compute the average two imprints as $I_{AVG1,2}(r, s) = I_1(r, s) + I_2(r, s)$ and the average of n imprints as $I_{AVGn}(r, s) = \frac{1}{n}\sum_{i=1}^{n} I_i(r,s)$.

For supporting blending between two imprints, we propose to use a transition function t(r, s) which has the same domain as the imprints being blended and range values $t(r, s) \in [0, 1]$. Two imprints can then be blended to form composed imprint $I_C(r, s)$

= $t(r, s)\cdot I_S(r, s) + (1 - t(r, s))\cdot I_T(r, s)$. A composed imprint can then be added to the target base imprint to facilitate pasting (Figure 10.3).

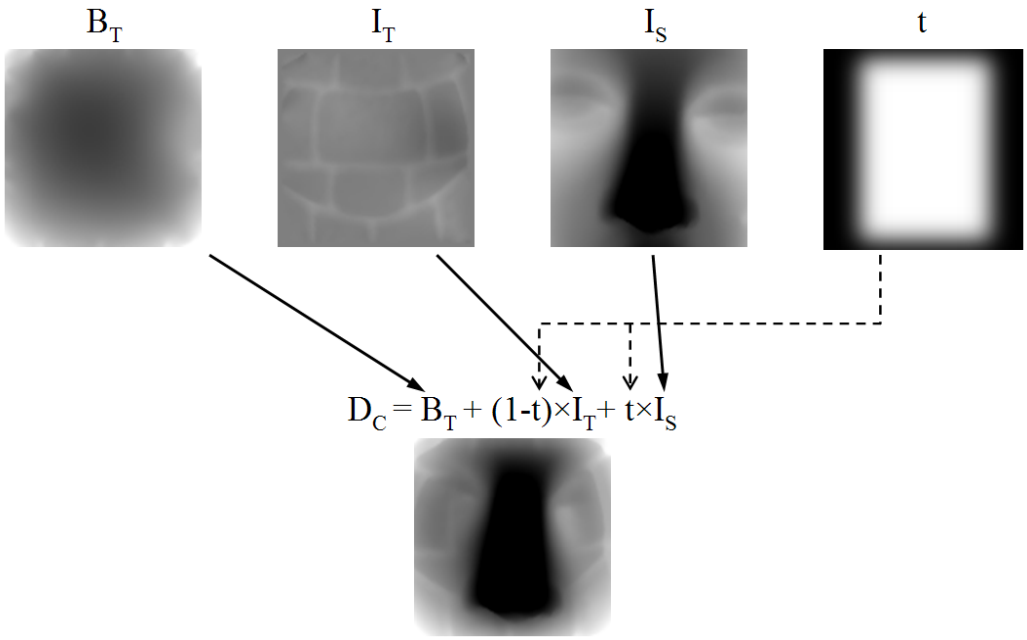

Figure 10.3: The result of blending a source and target imprint using a transition function is added to the target base surface to facilitate deposition.

10.5 Depositing

Similar to displacement mapping, pasting is achieved by perturbing the target surface geometry according to the desired feature shape. In our case, we deposit a feature by perturbing the surface given vertex positions computed from an imprint. In our implementation, each vertex of the mesh obtains its coordinate (r, s) in the camera (given the device parameters). If it is found to be in the camera view or, more specifically, the ROI, then it is a candidate for moving. It then obtains its 3D position by interpolating the depth value from the composed imprint (using bi-linear interpolation of the imprint

samples) and computing the 3D point from this result using the device parameters. We use the existing mesh connectivity; hence, there is no re-tesselation of the surface, though subdivision or re-sampling could be used to support more fidelity if desired, e.g. [Biermann et al. 2002; Turk 1992].

Our algorithm for adjusting the vertex positions based on an imprint is as follows. We first get the rendering parameters R from the OpenGL pipeline, where R consists of the 4x4 projection matrix, the 4x4 modelview matrix, and the 4x1 viewport parameters vector. For each vertex v_i in the ROI, we use R to project v_i to screen space (e.g. using gluProject()). This gives us imprint coordinates (r_i, s_i) as well as a z-buffer value z_i corresponding to v_i. The imprint coordinates (r_i, s_i) are continuous values while the imprint itself is represented as a discretely sampled image. Hence, we use interpolation to estimate the value of $D(r_i, s_i)$ from its neighbors. In our implementation, we use bicubic interpolation, which uses 16 neighbor samples to interpolate the value instead of 4 neighbor samples as in bilinear interpolation.

Once we have obtained $D(r_i, s_i)$, we use R to "un-project" (r_i, s_i, z_i) back into model space (e.g. using gluUnProject()), where $z_i=D(r_i, s_i)$. This gives us a coordinate in model space which we assign as the new position for vertex v_i. Results and comparisons of depositing relief features on surfaces with varying connectivities can be found in Figure 10.25 through Figure 10.29.

10.5.1 Feature scale

In our discussion of a rendering pipeline-based implementation of imprint scanning and depositing, we have thus far ignored the matter of scale model space feature scale. In our camera metaphor, the size of the imprint captured is normalized with respect to the

device frame. Then, the model space size of the imprint deposited is determined by the device frame. Hence, the relative transformation between the source and target device frames indicates the scaling that the feature undergoes as a result of the feature transfer process and this scaling is computed by construction. However, in our implementation, we need to account for the scaling explicitly.

We estimate the feature size by finding three points on the source or target surface which correspondingly project to three of the four corners of the imprint. The goal is to roughly estimate the model space width and height of the ROI orthogonal to the view direction. We compute approximate coordinates for these three points by the following process: (1) Pick an arbitrary pixel, e.g. the center pixel, of the imprint and obtain its z-buffer value z_c. (2) Use z_c with each of the three corners (r, s) = (0, 0), (w, 0), and (0, h) of the imprint along with R to obtain three points A, B, and C, respectively, in model space coordinates corresponding to the three corners we are using. (3) The model space dimensions of the imprint are then wsize=dist(A, B) and hsize=dist(A, C).

When we deposit an imprint, we compute a scale factor sf as the average ratio between the source and target feature sizes: $sf = 0.5 \cdot (wsizeT/wsizeS + hsizeT/hsizeS)$. The scale factor is then applied when composing offset imprint of the source with the target: $I_C(r, s) = sf \cdot t(r, s) \cdot I_S(r, s) + (1 - t(r, s)) \cdot I_T(r, s)$. The result of depositing using scale factor adjustment is illustrated in Figure 10.4, where the pasting location corresponds to the target region specified in Figure 10.1b.

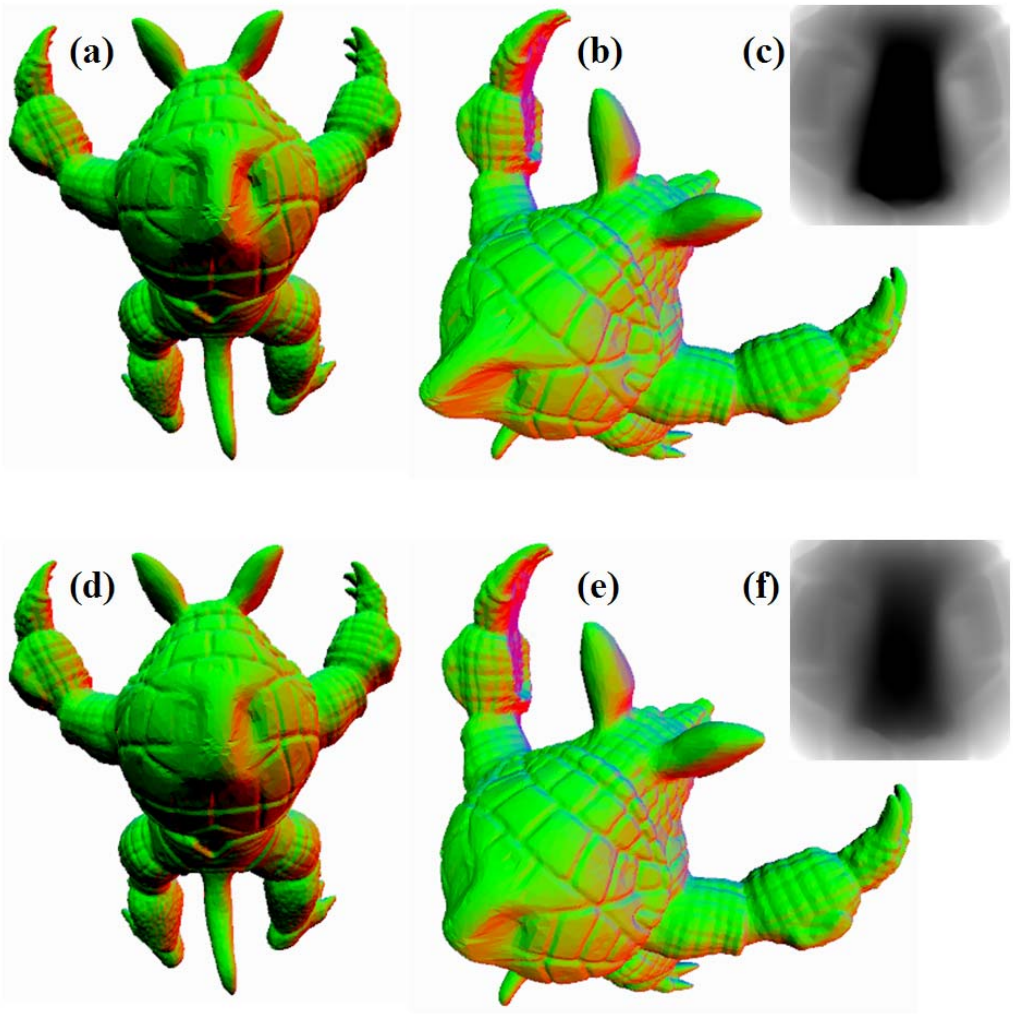

Figure 10.4: Result of depositing an imprint after scale factor adjustment (a, b, & c) and an extra 50% scale factor adjustment (d, e, & f). (a) and (d) give the original pasting views, (b) and (e) are alternate views better depicting the feature scale difference, and (c) and (f) are the depth imprints D_C deposited.

10.6 Results

We have described a prototype implementation of the basic components needed for imprint-based relief feature transfer. These components can then be used to facilitate the relief feature editing operations as described in Section 9.6. Here we provide additional results.

Figure 10.6, Figure 10.7, Figure 10.8, Figure 10.9, Figure 10.10, Figure 10.11, and Figure 10.12 give examples of imprint-based relief feature editing on the Stanford bunny model. Figure 10.5 shows the target selection box for all of the relief editing in this series of examples (on the Stanford bunny model). The imprint of "GT" lettering and the transition mask used throughout this series were created using image editing software.

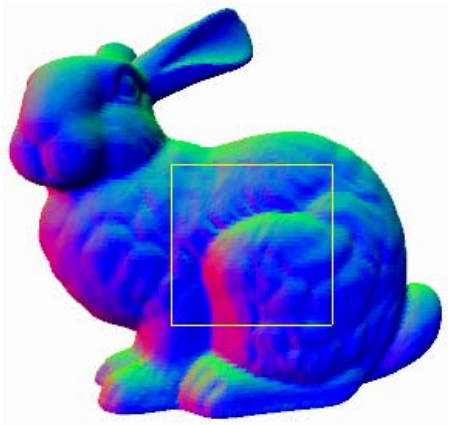

Figure 10.5: Target selection box for the bunny editing series.

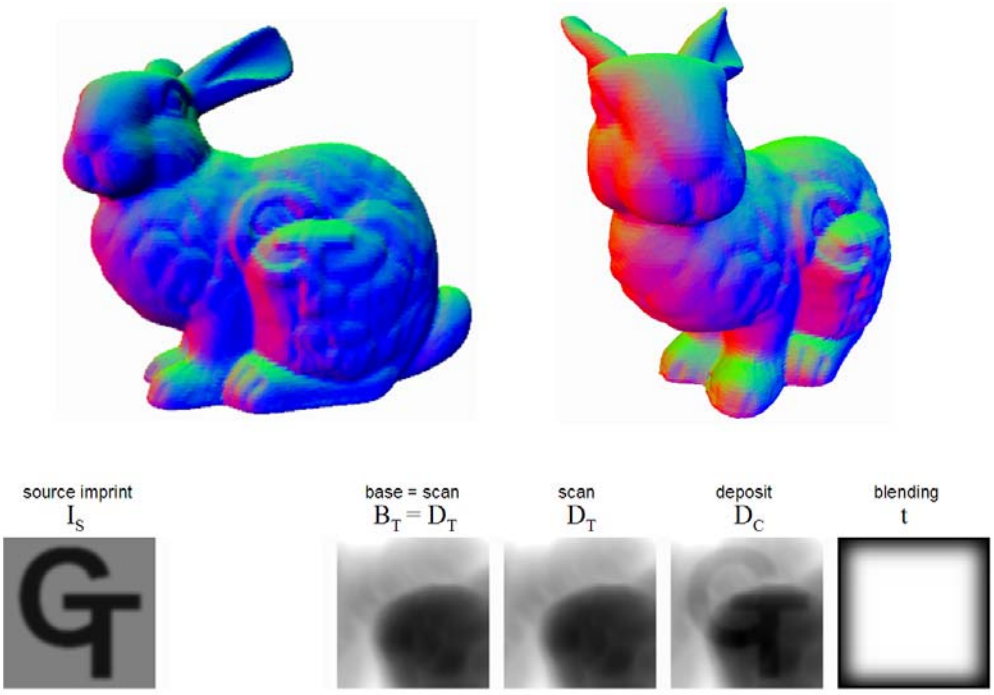

Figure 10.6: Pasting an imprint of protruding letters onto the original target surface.

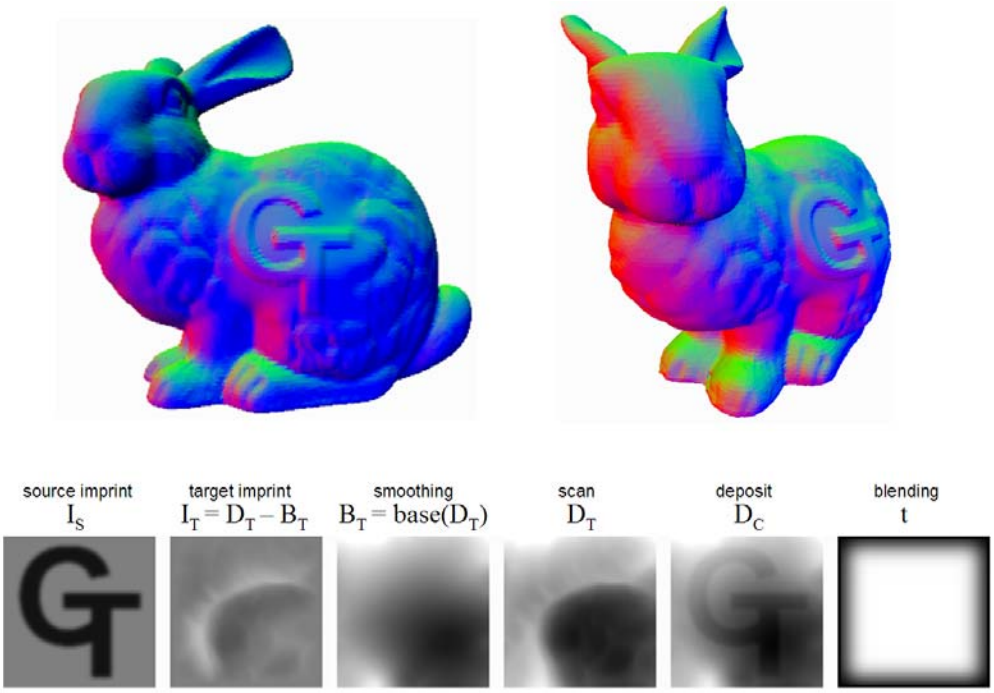

Figure 10.7: Pasting an imprint of protruding letters onto a smoothened base.

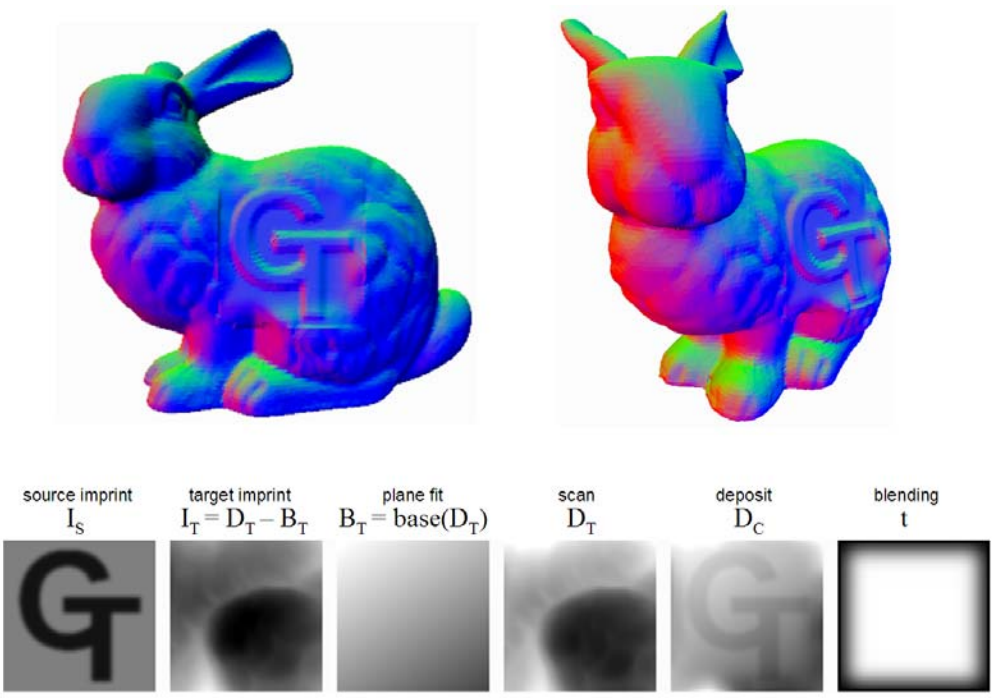

Figure 10.8: Pasting an imprint of protruding letters onto a plane fit base.

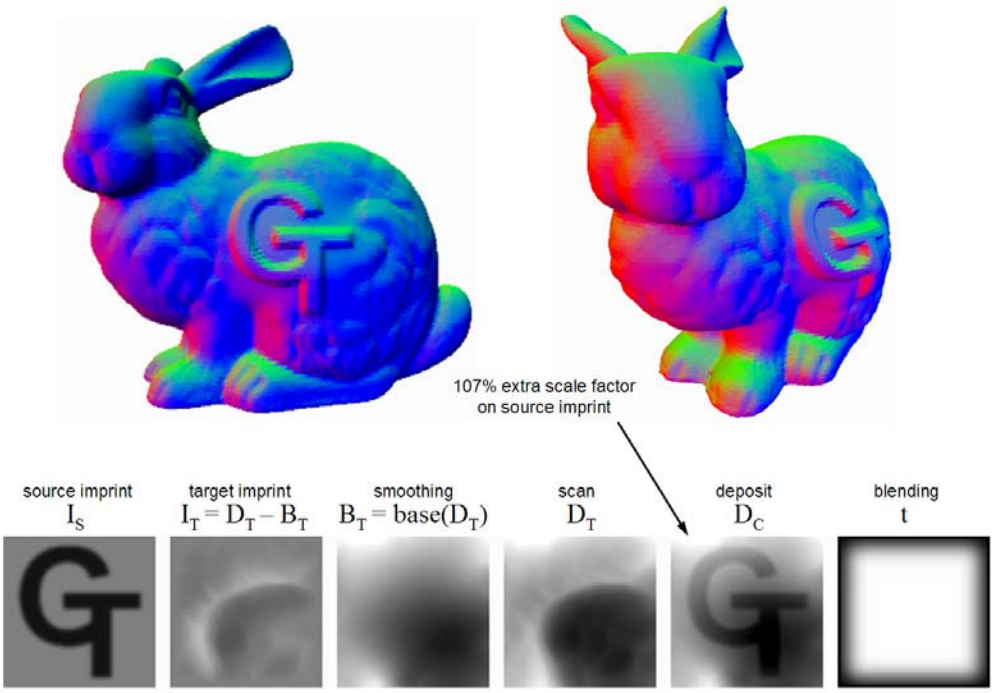

Figure 10.9: Pasting an imprint of protruding letters onto a smoothened base, where an extra 107% scale factor is applied to the source imprint before composing the final imprint to be deposited.

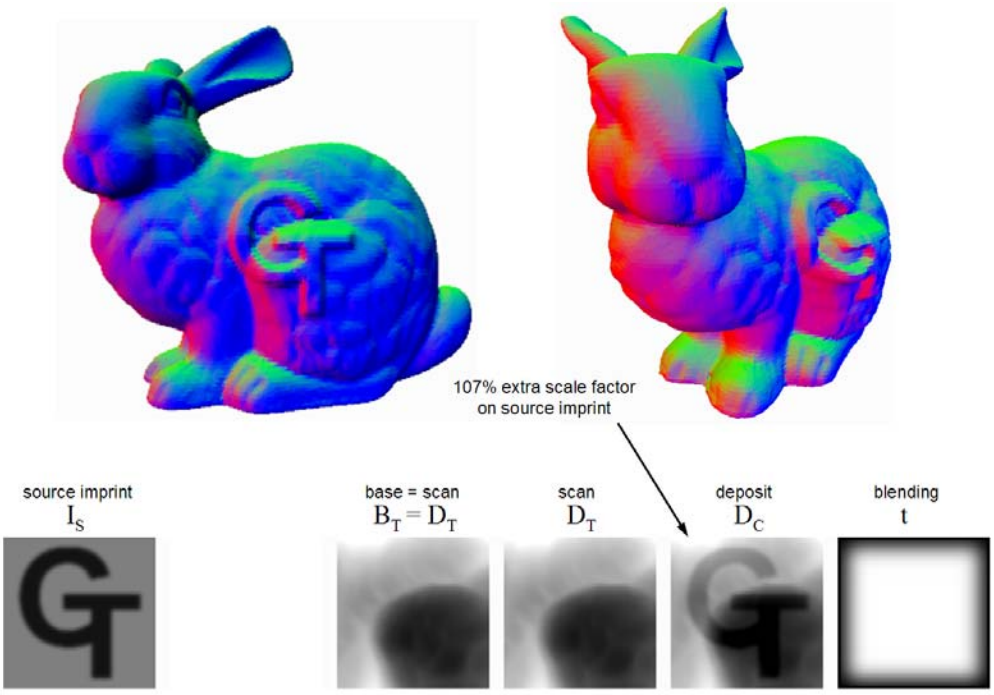

Figure 10.10: Pasting an imprint of protruding letters onto the original target surface, where an extra 107% scale factor is applied to the source imprint before composing the final imprint to be deposited.

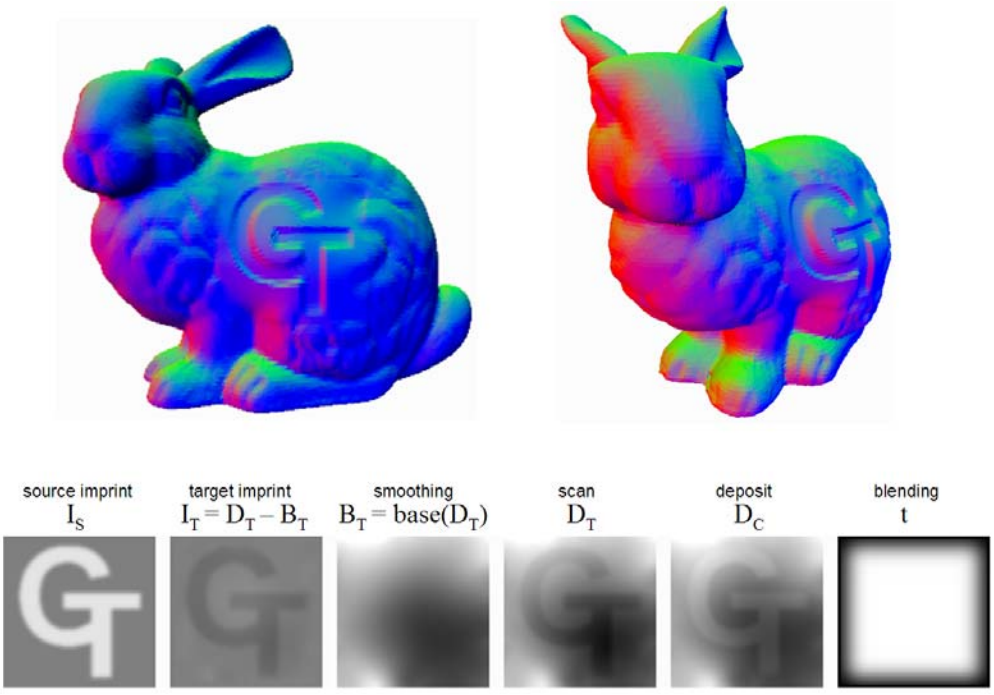

Figure 10.11: Consecutive pasting operations. Pasting an imprint of protruding letters followed by pasting an inward protruding version of the same imprint, both using smoothing to compute the base. The source surface is the same as the result in Figure 10.7.

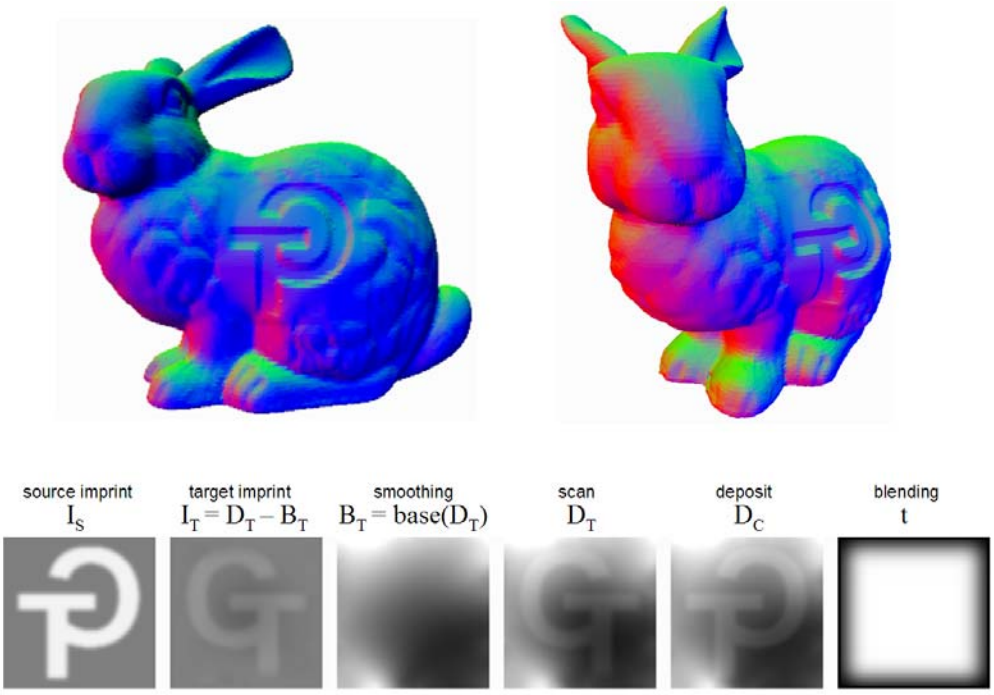

Figure 10.12: Consecutive pasting operations. The inward protruding imprint is mirrored and re-pasted on the result from Figure 10.11. Smoothing is used to compute the base surface.

If the target mesh is not sufficiently sampled, artifacts may appear (Figure 10.13 and Figure 10.14). Adaptive sampling may be attempted, but it may not recover the sharp edges of the copied feature, unless sharpen&bend subdivision [Attene et al. 2005] (or some similar process) is applied.

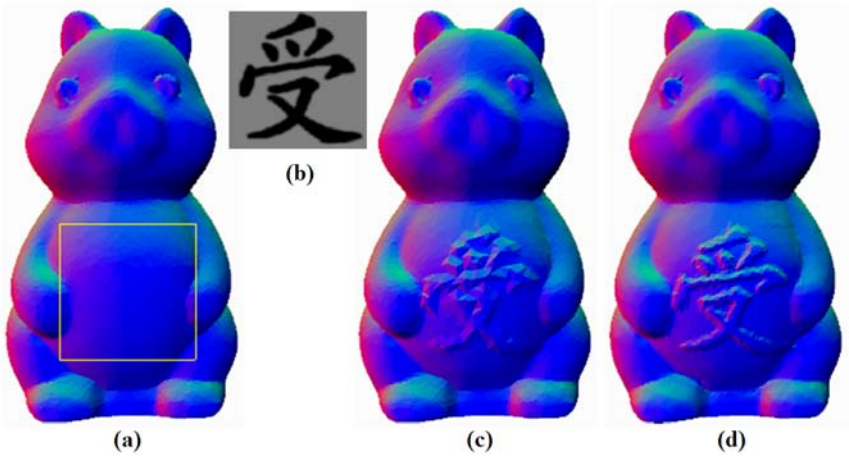

Figure 10.13: Pasting on surfaces with different sampling densities: (a) target ROI, (b) source imprint, (c) 20k triangle model, (d) subdivided 80k triangle model.

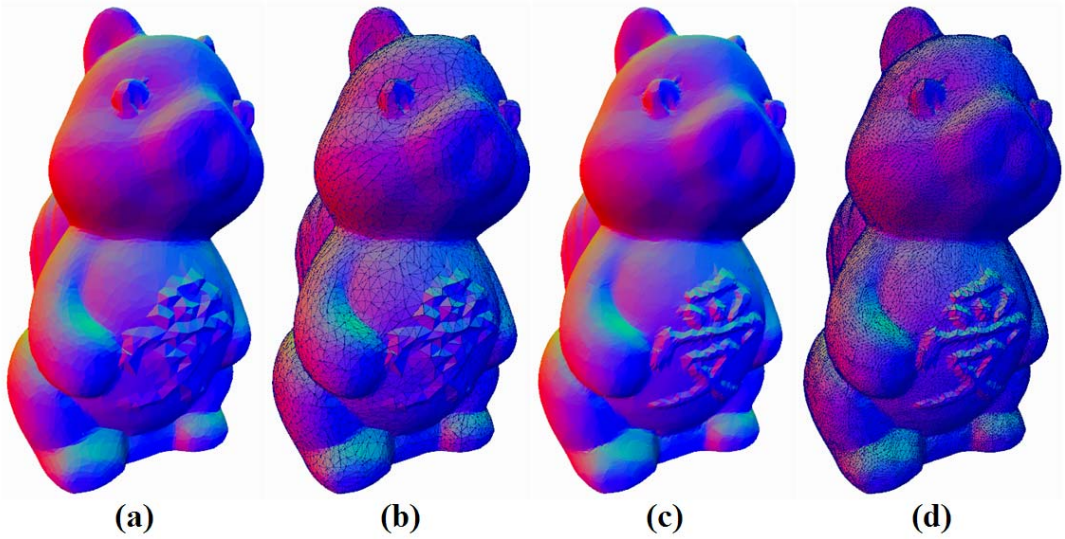

Figure 10.14: Alternate view and wireframes of pasting results on (a&b) a 20k triangle model and (c&d) a 80k triangle model (obtained via subdivision).

Copy, delete, cut, paste, and move operations can be illustrated using a single slide operation example. Figure 10.15 shows several steps in the process of sliding the left ear of the Stanford bunny model. The original model is shown in Figure 10.15a and Figure 10.15d; Figure 10.15b and Figure 10.15c show two different views of the model after the eye is cut (copy plus delete); and Figure 10.15e and Figure 10.15f show the model after

moving (cut plus paste) the eye to two different locations. Two different views of a side-by-side comparison of the model before and after sliding are given in Figure 10.16.

Note that the most time-consuming step in our prototype implementation of our sliding approach is the base smoothing. Our un-optimized implementation of 1000 iterations of smoothing takes two or three seconds to run on a single-processor 1.7GHz machine. However, this is not an inherent limitation of our imprint-based approach. More efficient algorithms exist for computing such a smooth base surface given boundary constraints [Farbman et al. 2009]. On the other hand, the base computation using plane-fitting which we implemented allows sliding to be achieved at interactive rates. Hence, a benefit of our approach is that a designer can actually slide the feature in realtime, as no local Boolean or topology fixing is needed.

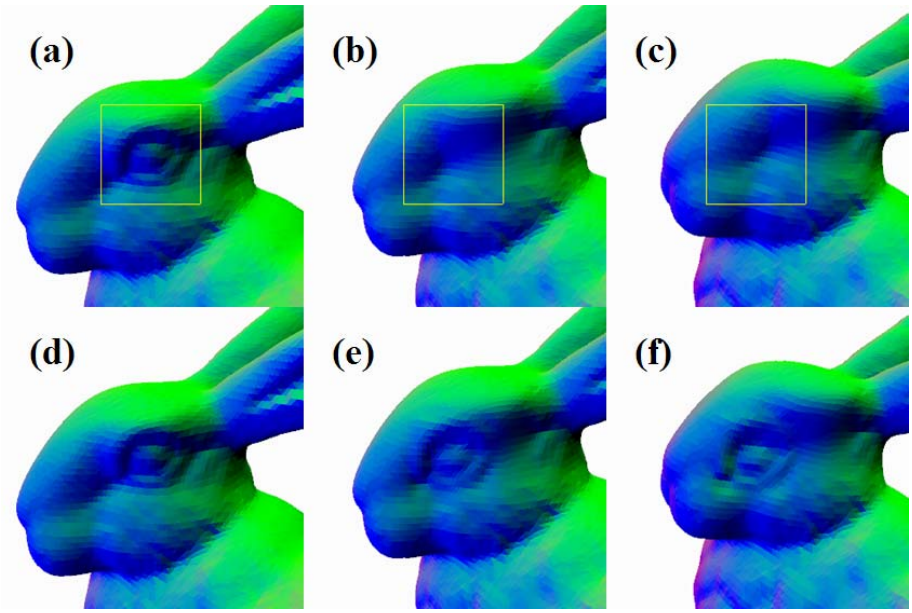

Figure 10.15: Sliding a relief. The eye of a bunny is slid from it original location (a and d) to an intermediate location (b and e) to its final position (c and f). The ROI selection box corresponding to the steps in d, e, and f are given in a, b, and c, respectively.

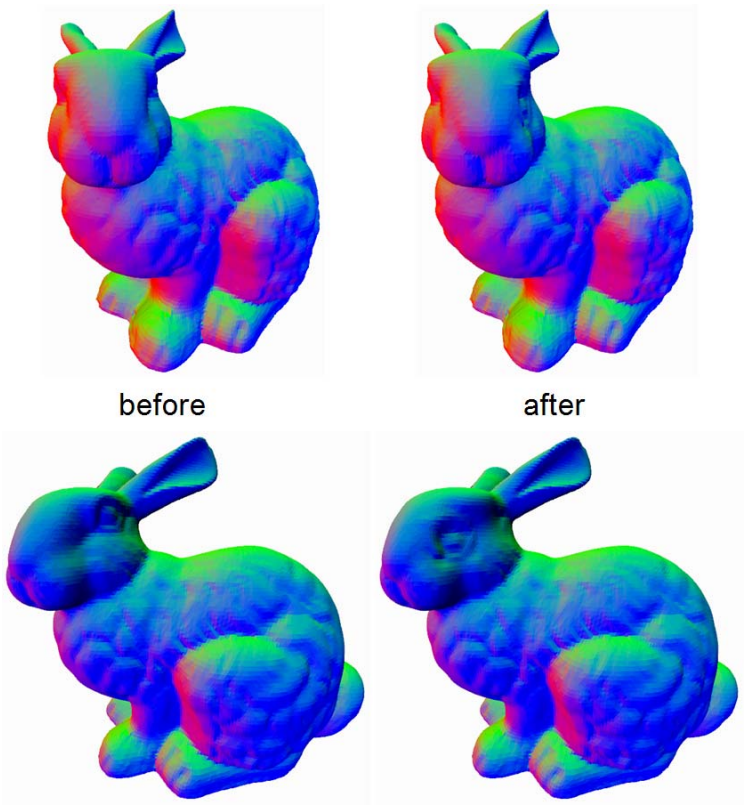

Figure 10.16: Two views comparing the bunny model before and after sliding its left eye forward.

Figure 10.17 gives an example of a pattern of relief features on an elephant model. In the example, a pattern of five relief feature instances is defined using a regular pattern of five frames. Figure 10.17b uses a flattened pyramid extracted from another model as the pattern leader. The pattern leader is changed in Figure 10.17c such that it uses the imprint of a diamond-shaped pyramid extracted from a different model.

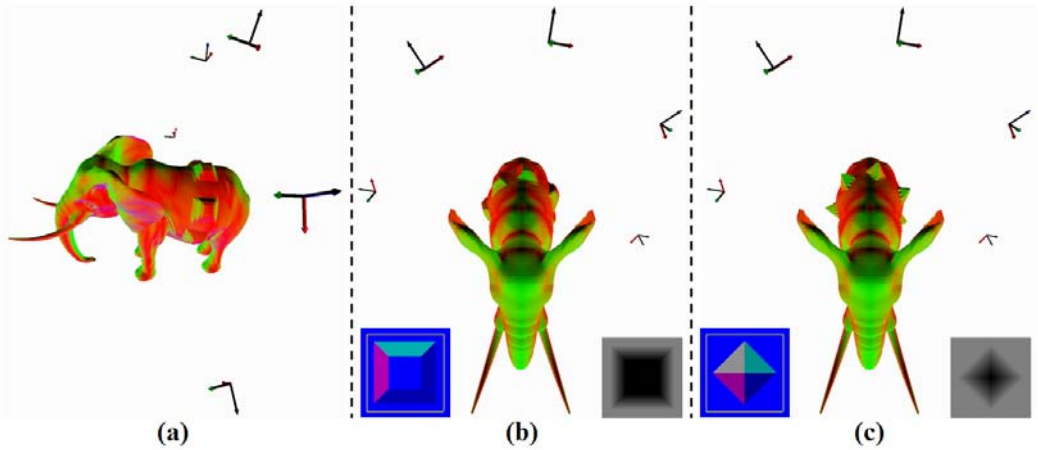

Figure 10.17: A regular pattern of relief features on an elephant model is defined using a regular pattern of frames (depicted as mini-axes). The pattern leader in (b) is a flattened square pyramid while the pattern leader in (c) is a diamond pyramid. The source models and respective imprints of the pattern leaders are shown to the left and right of the models in (b) and (c).

The pattern of five diamond-shaped pyramids in Figure 10.17c (and Figure 10.18a) is edited to include an exception in Figure 10.18b. Then, the pattern count is increased to seven in Figure 10.18c and the exception on the second instance is maintained.

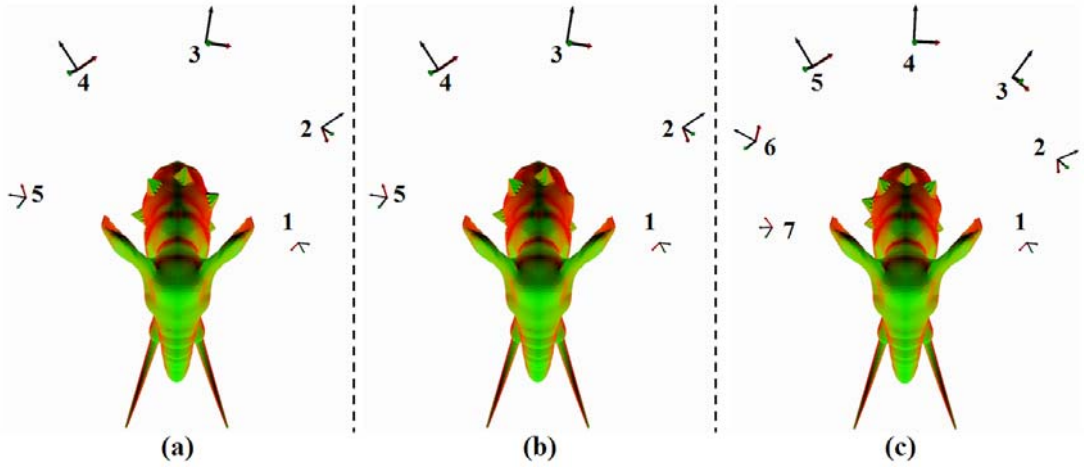

Figure 10.18: (a) Regular pattern of five relief instances on a surface. (b) An exception is defined where the imprint of the second instance is smoothed. (c) The pattern count is increased to seven and the exception is maintained.

Figure 10.19 shows a pattern of pattern of bump features on a sphere. Essentially, each device frame defines a separate projection. Hence, a pattern of device frames defines a collection of separate projections which are related by the pattern transform.

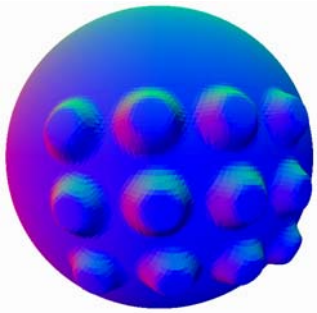

Figure 10.19: Pasting result of a pattern of pattern.

Finally, we show results from the modeling scenario in Figure 1.17. Figure 10.20 corresponds with Figure 1.17a and shows the original model with clean sharp edges. Figure 10.21 corresponds with Figure 1.17b and shows the model after recognition, base computation, removing all the features, and re-pasting one of the features in place. Figure 10.22 corresponds with Figure 1.17c and shows the entire 5x3 pattern re-pasted. Notice that the clean sharp edges are not retained due to the sampled imprint which can only approximate the relief shape to pixel-level accuracy. Also, from a single view, certain edges (e.g. around the window outside frame) overlap other parts of the surface (Figure 1.17f and Figure 1.17g). Figure 10.23 corresponds with Figure 1.17d and shows an exception on the second row where all the windows are rotated and slightly shrunk. Figure 10.24 corresponds with Figure 1.17e and shows the result after parametric editing where the horizontal count is changed to 4 and the horizontal spacing is increased. Notice that the exceptions are maintained. Figure 10.25 shows a close-up of window (3, 1) from

Figure 10.20 (the original façade model). Figure 10.26 shows a close-up of window (3, 1) from Figure 10.22 (the pattern re-pasted in place). Figure 10.27 shows a close-up of window (3, 2) from Figure 10.23 where the window is rotated and shrunk in its original location. Figure 10.28 shows a close-up of window (3, 2) from Figure 10.24 where the window is rotated and shrunk and has a new position on the facade away from its original location. Figure 10.29 shows a close-up of window (3, 1) from Figure 10.24 where the window has a new position on the facade away from its original location. This series of close-ups show how the underlying sampling (Section 10.5) can affect the quality of the reconstruction.

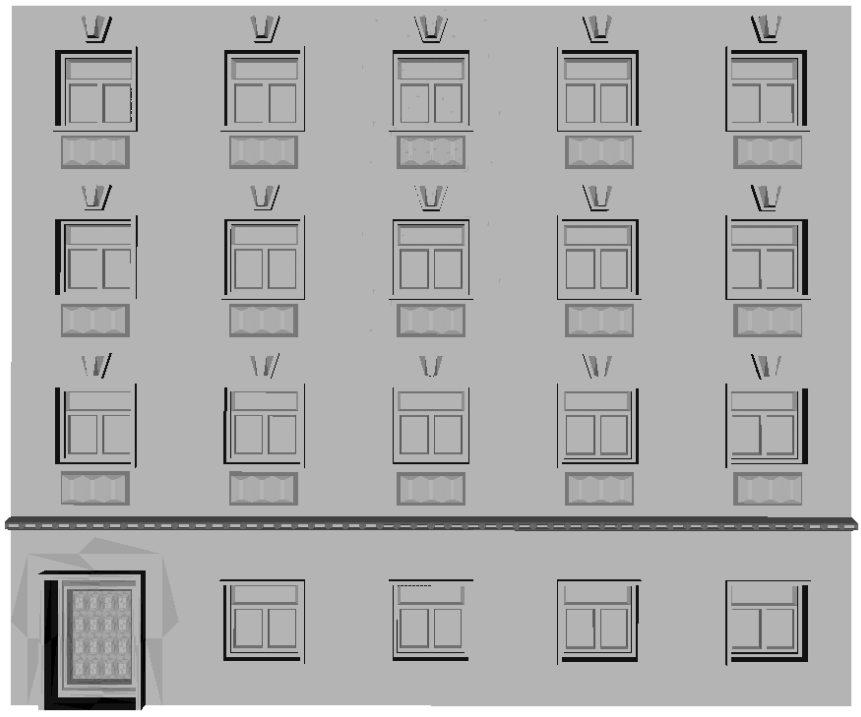

Figure 10.20: Original input facade model.

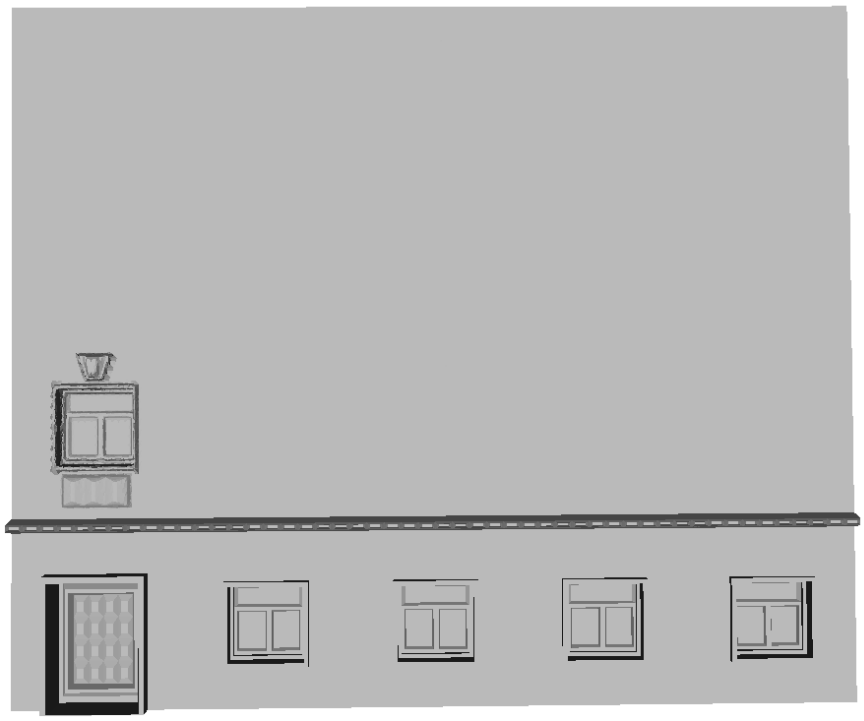

Figure 10.21: Facade model after recognition, base computation, feature removal, and re-pasting one instance.

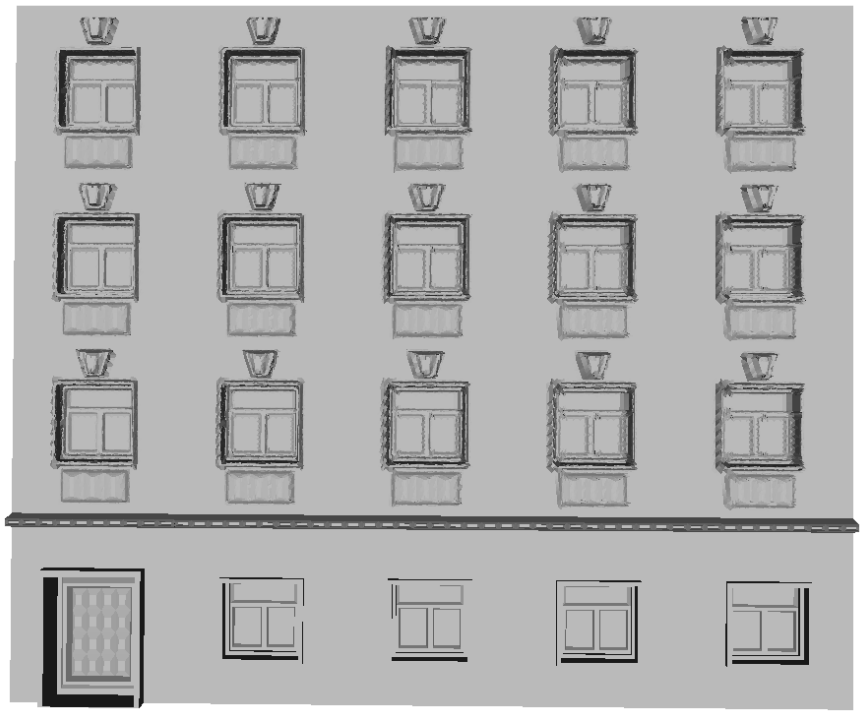

Figure 10.22: Facade model after re-pasting the entire 5x3 pattern of windows.

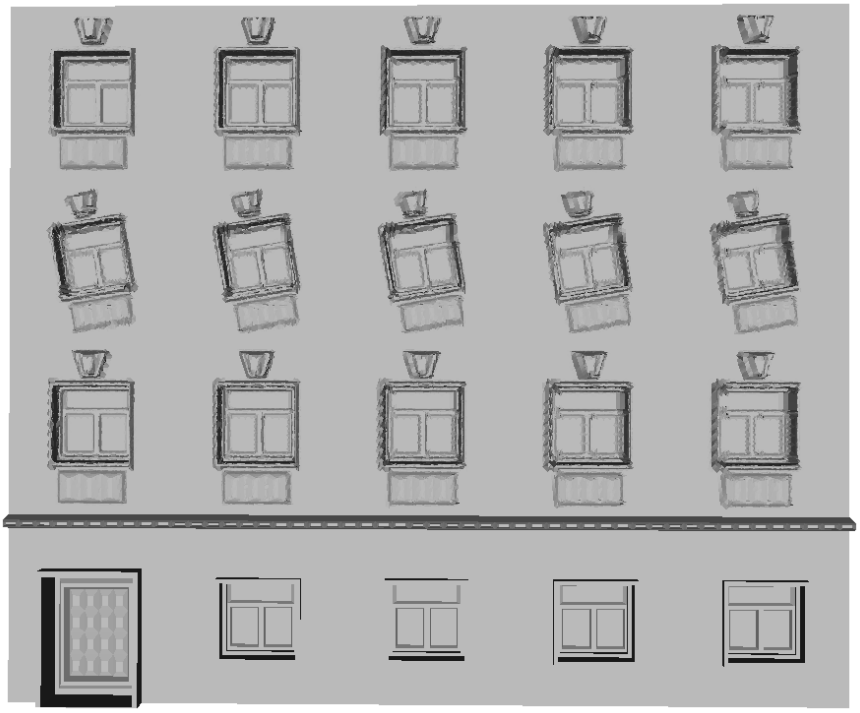

Figure 10.23: Facade model after second row exception where the windows are rotated and shrunk.

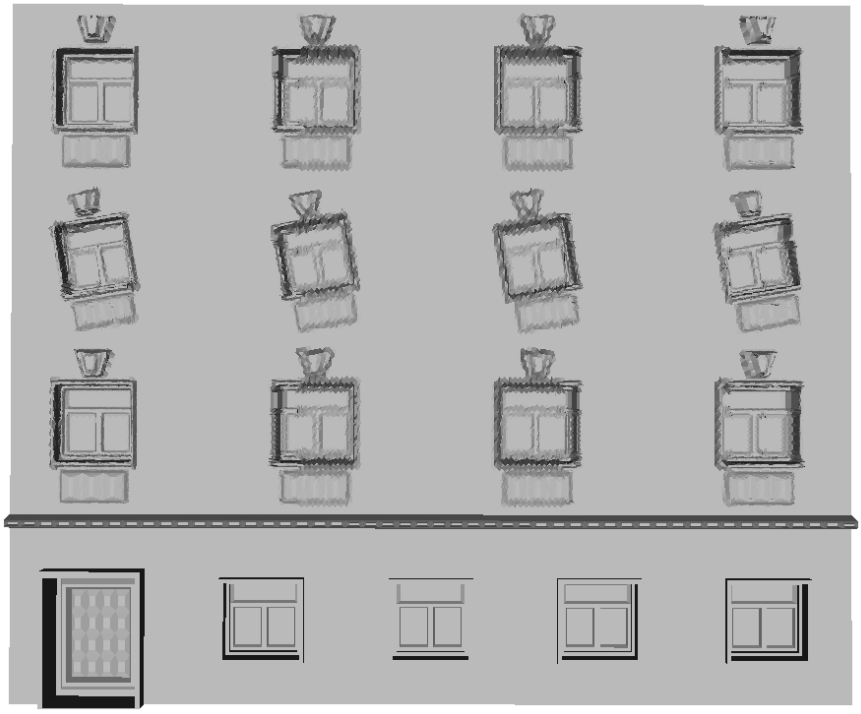

Figure 10.24: The facade retains the exceptions after changing pattern paramters.

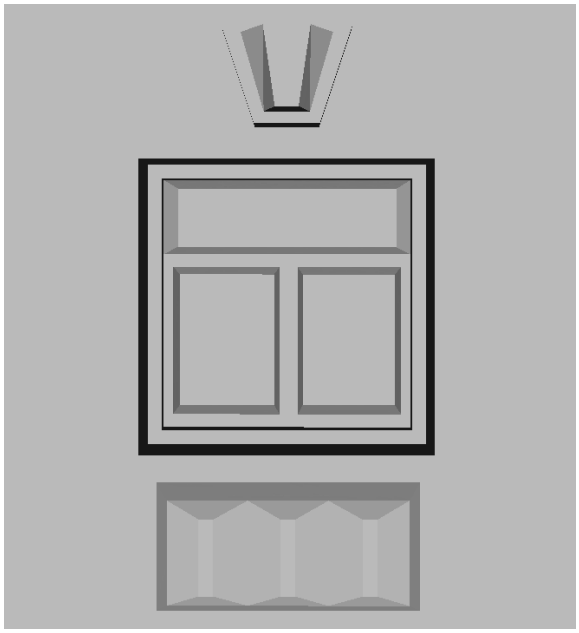

Figure 10.25: Close-up view of window (3, 1) on the original facade.

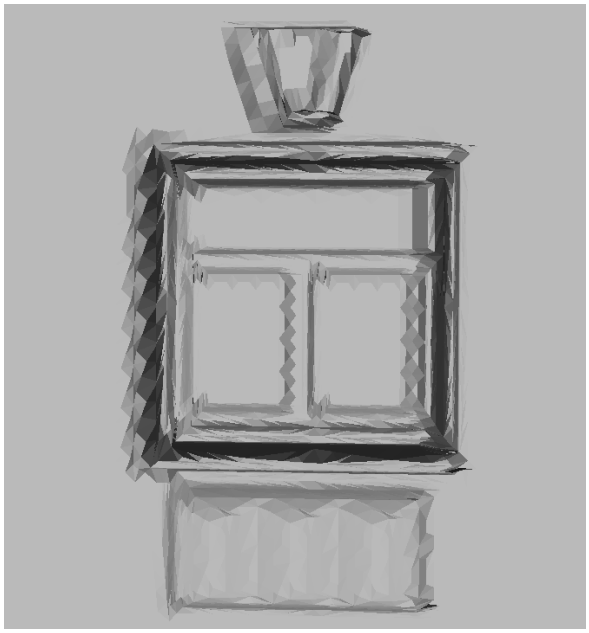

Figure 10.26: Close-up view of window (3, 1) after re-pasting.

Figure 10.27: Close-up view of window (3, 2) after being rotated, shrunk, and re-pasted.

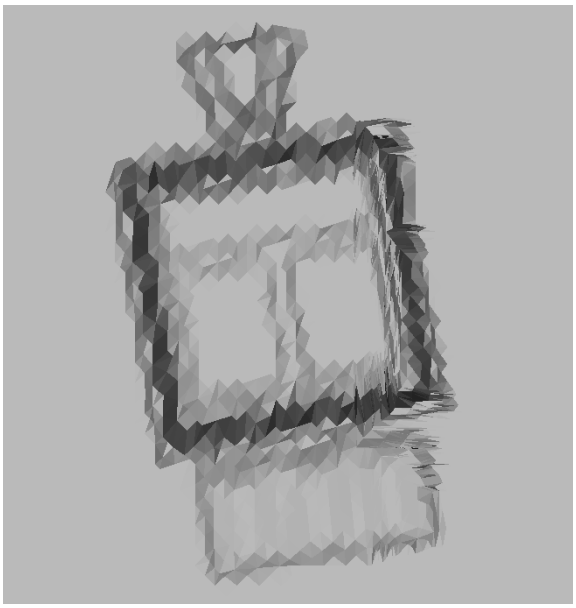

Figure 10.28: Close-up view of window (3, 2) after being rotated, shrunk, re-positioned, and re-pasted.

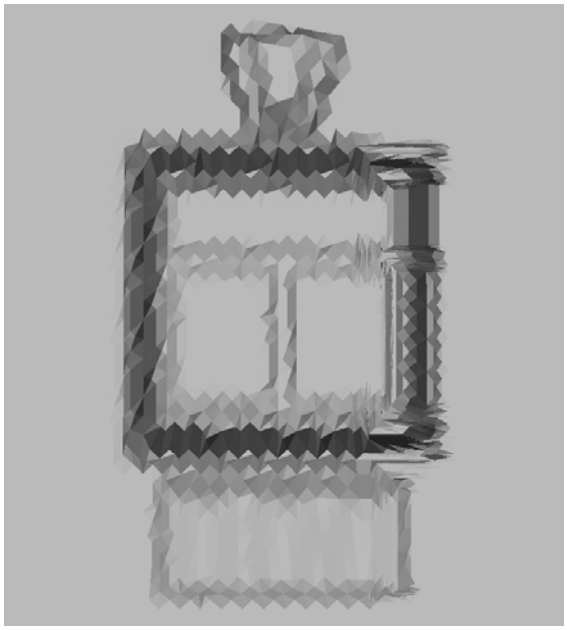

Figure 10.29: Close-up view of window (3, 1) after being re-pasted in a new position.

10.7 Discussion

Here we give further discussion on several aspects of our imprint-based relief editing approach: representation independence, imprint space editing, and target surface preservation.

10.7.1 Representation independence

We may say that our approach is representation independent. Because most of the processing is performed on the scanned imprint data, we are able to design any number of relief editing operations without being constrained by whether the original surface representations are compatible. Even the scanning step only depends on being able to rasterize the geometry into the depth buffer – technology available for many representation types including parametric shapes, subdivision surfaces, NURBS, and unstructured triangle meshes. Note that only the scan and deposit operations, i.e. the

interface between imprint space and a surface in world space, are representation dependent. All of the relief processing is performed in imprint space. This includes base computation (both reference and replacement), encoding, modification, mixing, and composition. Hence, as long as scan and deposit operations can be implemented for a given representation, that representation can participate in the relief feature information passing and imprint-based relief editing made available through imprint space.

10.7.2 Imprint space editing

An imprint-based approach also gives us a flexible handle on editing relief features. For instance, imprint space processing can leverage image processing algorithms and techniques, which are numerous, usually relatively straightforward to implement, and allow for efficiency improvements such as parallelization. Also, there is the possibility of allowing the designer to use a familiar image editing interface to process reliefs. For example, the designer can create embossed lettering or custom design the desired base surface using a grayscale image to represent a height field. In addition, imprints allow us to compose multiple relief features using simple image addition, subtraction, and averaging operations. We can just as easily morph between features using the same operations.

10.7.3 Target surface preservation

Our imprint-based relief editing approach also allows both the connectivity and geometry of the target surface to be preserved. For instance, in our implementation of depositing based on a triangle mesh, each existing vertex in the ROI queries the imprint to determine its new perturbed position. The designer may optionally adjust the

connectivity, for example, by subdividing the mesh locally, to be able to receive and represent the shape information contained in the imprint with higher fidelity.

In a slide operation, preservation of surface geometry on the slide path is built-in to the approach. This is achieved using a modified paste operation which saves a copy D_R of the scanned imprint before pasting. Then, the portion of the surface that is currently supporting the pasted feature is restored using D_R before the feature is re-pasted in its next location on the slide path. Note that small numerical errors may be produced along the slide path due to the depth buffer sampling. We imagine that while carrying out an interactive slide operation, such numerical errors would not crucial to the designer experience since they are typically small enough not to be detected. However, in the case of certain shading methods, e.g. specular shading, small perturbations may affect the surface normal such that the change in resulting shading is noticeable. Hence, for producing the final result of a slide operation, we recommend that the original geometry be saved before sliding begins and that it be restored before pasting the feature at its final slide destination.

10.8 Summary of imprint-based relief editing

We have presented an approach for sliding geometric relief features on surfaces. Our approach offers the following benefits:

- The approach is simple to implement, not requiring the composition and integration of a collection of individual representation-dependent approaches to solve the various sub-problems involved in the process of relief feature transfer.

- The approach is representation independent in the sense that the way it extracts the relief, processes relief information, and composes and reconstructs the surface shape to be pasted does not depend on the representation of the source or target surfaces.

- The source and target surface representations are not required to be the same. Furthermore, the source region (or the target region) itself can be composed of geometry from different surfaces and different representations.

- The approach allows the connectivity at the target to be preserved. In fact, the target sampling can be of arbitrary scale and density, both as compared with the source data (or the sampling of the relief extracted from the source) and across the target surface itself. This gives the designer the option of adjusting the tessellation of the connectivity at the target as required or desired.

- The approach is essentially a general feature transfer approach and hence also supports copy, cut, paste, and delete operations on single features. We believe it is also feasible to extend the basic algorithm into a framework which supports multiple feature sets including regular patterns and leave this to future work.

Our approach has the following limitations:

- The approach supports geometric reliefs on surfaces which are height fields locally in the direction of the relief capture.

- The approach assumes that the sampling density of the connectivity at the target is fine enough to receive and express the source feature detail, at least to the desired fidelity. If the sampling is not fine enough, artifacts may appear.

We suggest the following items for future work:

- Explore approaches to support automation in the pattern recognition (registration) process. Proper registration of the relief features in a pattern is essential for producing high quality results when processing relief features (e.g. when removing and re-pasting). This is especially important for a multiple instance operation such as beautification which relies on accurate alignment between all the feature instances identified by frames. We expect that image registration approaches, e.g. [Zitová and Flusser 2003], could be employed to assist feature alignment.

- Explore approaches to frame regularization for patterns of relief features where the pattern arrangement follows the surface. This is more challenging that frame regularization for a set of "free-floating" frames in space since the arrangement depends on the surface shape. Future work would define frame regularity over a surface and provide a method for producing regular frame arrangements over a surface.

- Explore applying the imprint approach to animating and morphing relief features. We may even consider an "animated relief" as a new type of feature. Due to our framework, this essentially becomes a problem of animating and morphing images. Future work may also explore combining this idea with our approach for sliding relief features over a surface to enable a more general surface relief animation approach.

- Most of the relief processing computation takes place in imprint space, a regular parametric space comparable to that of a 2D image. Hence we would like to explore ways of editing relief features which leverage existing image processing algorithms and even existing image processing tools. For example, image morphology operations and blurring operations could be applied to generate a custom blending function to pasting a relief feature and image despeckle or de-noising operations could be used to remove noise from

geometric reliefs. This idea of leveraging image processing for surface editing is powerful since there are a wide variety of existing tools and efficient algorithms for image processing, many of which are widely available and even familiar to many potential designers, not just those with expertise in working with 3D geometry.

11 CONCLUSION

We propose a clear and practical definition for regular patterns in the context of parametric modeling. Our definition distinguishes two semantically separate components of a pattern: (1) geometric content of a feature and (2) spatial arrangement of features. The spatial arrangement is given as a series of positions and orientations, which we call frames. Corresponding to these two components, our definition distinguishes two independent forms of regularity: (1) feature regularity and (2) frame regularity. Hence, a pattern may be feature regular without being frame regular, frame regular without being feature regular, irregular (neither feature regular nor frame regular), or regular (both feature regular and frame regular).

In line with our definition for regular patterns in the context of parametric modeling, we propose a modeling setting which encompasses procedures for converting a pattern between the various forms of regularity. The modeling setting also includes general editing of regular patterns such as modifying the pattern leader, adjusting the regular spacing/transformation, and changing the instance count. In addition, the modeling setting defines an additional state of regularity – regular with exceptions.

In PART I, we describe a hierarchical scene graph representation for regular patterns. Our representation is essentially a graph which defines the spatial arrangement of the components in a scene or assembly. The specific semantics encoded in the graph include simple regular patterns, grouping, nesting, and recursive nesting of patterns.

We then propose an approach for making selections of subsets of the feature instances arranged by a hierarchical pattern graph. Since the pattern graph reflects a form of

designer intent, we propose to use the information found in the structure and relational semantics encoded in the graph as natural guidelines for making selections of multiple components in a model. We propose one method for subset selection in models containing such relational semantics as we have described and do not claim to provide the best or only approach. However, the proposed scheme directly follows the semantic relations encoded in the graph and thus has intrinsic value. Such natural selections can provide a basis for more complex selections.

We show that these natural selections are also efficient to compute and store. Furthermore, the approach supports a scheme where the designer can interactively refine a selection, where selection guides can assist the designer in picking instances, and where the designer can explore the underlying relational semantics of the model. Such user interaction factors have the potential to increase productivity and efficiency of design and editing of CAD models containing patterns.

In PART II, we propose an approach for editing relief features on surfaces. Our main contribution is a set of algorithms to copy, delete, cut, paste, move, and slide relief features on a triangle mesh without re-sampling or modifying the connectivity of the mesh. As a theoretical base for our algorithms, we propose a practical definition of relief features as the difference between a surface and a theoretical base surface for a given domain of sampling rays. Based on this definition, we describe a general approach to identify, extract, process, and apply relief features on surfaces. While we demonstrate the approach using triangle meshes, the approach is not inherently dependent on a particular surface representation, hence, it can be considered representation-independent. In particular, the process of computing a base surface, separating the relief feature

information, and editing the shape of a relief feature is accomplished in imprint space, a space describing a scalar function with a two-dimensional domain corresponding to the domain of sampling rays. Practically, imprint space can be represented as a two-dimensional image and hence it can be stored, transmitted, edited, and processed like an image and using image processing algorithms and tools.

Our practical definition for relief features (presented in PART II) allows relief features to fit into the general framework of regular patterns (presented in PART I). Specifically, source/target specification can be achieved by defining a frame with respect to a surface; that is, we interpret a frame as a virtual camera that can "see" a particular portion of the surface. Hence, a single relief feature can be positioned and instantiated by defining a frame with respect to a surface. Likewise, a set of relief features can be arranged by defining an arrangement of frames with respect to a surface. Hence, as in the building façade example described in the Introduction (Section 1.4), this arrangement of frames can be represented as a pattern of frames having a regular arrangement in space. Given the set of frames, the geometry of the relief features can be edited or the frames can be re-arranged and the features re-instantiated according to the general pattern editing framework.

We identify several areas for future work.

Firstly, editing of regular patterns of relief features on surfaces depends on two main components: (1) the ability to instantiate a relief feature and (2) the ability to define regular arrangements of frames on a surface. We addressed instantiation, including the ability to copy, remove, cut, and paste relief features on surfaces. We addressed arrangement by allowing the frame regularity to be defined with respect to

transformations in space. As future work, we would like to explore ways of defining regular arrangements of frames which more closely follow the target surface.

Then, given a clear definition and approach for regular arrangement of frames on surfaces, future work may determine how to perform frame regularization of patterns of relief features on surfaces. Given an identified near-regular pattern, a frame regularization operation would regularize the arrangement of the frames with respect to the given definition of regular arrangement of frames on surfaces.

Finally, future work may include the determination of how to identify or recognize an existing pattern. Recognition in general is a challenging problem and much work has been presented which addresses the problem in the context of reverse engineering and shape analysis. As future work, we would like to investigate the possibility of performing automatic recognition of nearly regular patterns of relief features on surfaces.

REFERENCES

[3dsMax] 3ds Max 3D Animation and Rendering Software. Autodesk, Inc. http://www.autodesk.com/3dsmax. Sept. 28, 2009.

[Abelson and diSessa 1982] H. Abelson and A. A. diSessa. Turtle geometry. M.I.T. Press, Cambridge, 1982.

[Alexa 2002] M. Alexa. Linear combination of transformations. ACM Trans. Graph. 21, 3 (Jul. 2002), 380-387. 2002.

[Alliez et al. 2003] P. Alliez, É. C. Verdière, O. Devillers, and M. Isenburg. Isotropic Surface Remeshing. In Proceedings of the Shape Modeling international 2003 (May 12 - 15, 2003). Shape Modeling International. IEEE Computer Society, Washington, DC, 49. 2003.

[Attene et al. 2005] Marco Attene, Bianca Falcidino, Michela Spagnuolo, and Jarek Rossignac. Sharpen&Bend: Recovering curved edges in triangle meshes produced by feature-insensitive sampling. IEEE Transactions on Visualization and Computer Graphics (TVCG), vol 11, no 2, pp. 181-192, March/April 2005.

[Attene etal. 2006] Marco Attene, Sagi Katz, Michela Mortara, Giuseppe Patan, Michela Spagnuolo and Ayellet Tal. Mesh Segmentation: A Comparative Study. In Proceedings of Shape Modelling International (SMI'06), IEEE Computer Society Press, pp. 14-25, 2006.

[Attene etal. 2008] Marco Attene, Michela Mortara, Michela Spagnuolo and Bianca Falcidieno. Hierarchical Convex Approximation of 3D Shapes for Fast Region Selection. Computer Graphics Forum, Vol. 27, No. 5 (SGP'08 Procs.), pp. 1323-1333, 2008.

[AutoCAD] AutoCAD. Autodesk, Inc. http://www.autodesk.com/autocad. Sept. 28, 2009.

[Bajaj et al. 1995] C. L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques S. G. Mair and R. Cook, Eds. SIGGRAPH '95. ACM, New York, NY, 109-118. 1995.

[Barequet and Sharir 1995] G. Barequet and M. Sharir. Filling gaps in the boundary of a polyhedron. Computer Aided Geometric Design 12, 2, 207-229. 1995.

[Barghiel et al. 1995] C. Barghiel, R. Bartels, and D. Forsey. Pasting Spline Surfaces. Mathematical Methods for Curves and Surfaces, Vanderbilt University Press, 31-40, 1995.

[Barnett and Lewis 1994] V. Barnett and T. Lewis. Outliers in Statistical Data. John Wiley & Sons., 3rd edition. 1994.

[Benko et al. 2001] P. Benko, R. R. Martin, and T. Virady. Algorithms for reverse engineering boundary representation models. Computer-Aided Design, Volume 33, Issue 11, 14 September 2001, Pages 839-851. 2001.

[Biermann et al. 2002] Henning Biermann, Ioana Martin, Fausto Bernardini, Denis Zorin. Cut-and-Paste Editing of Multiresolution Subdivision Surfaces. ACM Transactions on Graphics, vol 21, Number 3, Proceedings of ACM Siggraph, pages 312-321, July 2002.

[Branch et al. 2006] John Branch, Flavio Prieto, and Pierre Boulanger. A Hole-Filling Algorithm for Triangular Meshes Using Local Radial Basis Function. Proceedings, 15th International Meshing Roundtable, Springer-Verlag, pp.411-432, September 17-20, 2006.

[CATIA] CATIA. Dassault Systems. http://www.3ds.com/products/catia/. Sept. 28, 2009.

[Cenani and Cagdas 2007] S. Cenani, G. Cagdas. A Shape Grammar Study: Form Generation with Geometric Islamic Patterns. GA2007: Generative Art 10th International Conference, Politecnico Di Milano University, Milano, Italy, December 2007, 216-222. 2007.

[Chan et al. 1997] Leith Kin Yip Chan, Stephen Mann, and Richard Bartels. World space surface pasting. Proceedings of Graphics Interface, pp. 146-154. 1997.

[Chen etal. 2009] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A Benchmark for 3D Mesh Segmentation. ACM Transactions on Graphics (Proc. SIGGRAPH). 28(3) August 2009.

[Clarenz et al. 2004] U. Clarenz, U. Diewald, G. Dziuk, M. Rumpf, and R. Rusu. A finite element method for surface restoration with smooth boundary conditions. Comput. Aided Geom. Des. 21, 5 (May. 2004), 427-445. 2004.

[Conrad and Mann 1999] B. Conrad and S. Mann. Better Pasting Via Quasi-Interpolation. Curve and Surface Design: SaintMalo, Vanderbilt University Press, 27-36. 1999.

[CorelDRAW] CorelDRAW. Corel Corporation. http://www.corel.com/. Sept. 28, 2009.

[Curless and Levoy 1996] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, Annual Conference Series, 303-312. 1996.

[Davis et al. 2002] J. Davis, S. R. Marschner, M. Garr, and M. Levoy. Filling holes in complex surfaces using volumetric diffusion. In Proceedings of the 1st International Symposium on 3D Data Processing Visualization and Transmission (3DPVT-02),

IEEE Computer Society, Los Alamitos, CA, G. M. Cortelazzo and C. Guerra, Eds., 428-438. 2002.

[Desbrun et al. 1999] M. Desbrun, M. Meyer, P. Schroeder, and A. H. Barr. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In Proceedings of SIGGRAPH 99, 317-324. 1999.

[Desbrun et al. 2002] M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. Proceedings of Eurographics '02, pp. 209-218. 2002.

[Dyken and Floater 2009] C. Dyken and M. Floater. Transfinite mean value interpolation. Computer Aided Geometric Design, Volume 26, Issue 1, January 2009, Pages 117–134. 2009.

[Ebert et al. 2003] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley. Texturing & modeling a procedural approach. 3rd Ed. Morgan Kaufmann, San Francisco, CA. 2003.

[Eck et al. 1995] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques S. G. Mair and R. Cook, Eds. SIGGRAPH '95. ACM, New York, NY, 173-182. 1995.

[Farbman et al. 2009] Z. Farbman, G. Hoffer, Y. Lipman, D. Cohen-Or, and D. Lischinski. Coordinates for instant image cloning. In ACM SIGGRAPH 2009 Papers (New Orleans, Louisiana, August 03 - 07, 2009). H. Hoppe, Ed. SIGGRAPH '09. ACM, New York, NY, 1-9. 2009.

[Floater 1997] M. S. Floater. Parametrization and smooth approximation of surface triangulations. Comput. Aided Geom. Des. 14, 3 (Apr. 1997), 231-250. 1997.

[Floater 2003] M. S. Floater. Mean value coordinates. Computer Aided Geometric Design 20, 1 (Mar. 2003), 19-27. 2003.

[formZ] form-Z. AutoDesSys, Inc. http://www.formz.com/. Sept. 28, 2009.

[Fu et al. 2004] Hongbo Fu, Chiew-Lan Tai, and Hongxin Zhang. Topology-free cut-and-paste editing over meshes. Proc. 3rd Int. Conf. Geometric Modeling and Processing (GMP 2004), IEEE, Apr 2004.

[Funkhouser et al. 2004] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by Example. ACM Transactions on Graphics (SIGGRAPH 2004), Los Angeles, CA. 652-663. 2004.

[Ganster and Klein 2007] Björn Ganster and Reinhard Klein. An Integrated Framework for Procedural Modeling. In proceedings of Spring Conference on Computer Graphics 2007 (SCCG 2007), pages 150-157, Comenius University, Bratislava, Apr. 2007.

[Gao et al. 2003] C. H. Gao, F. C. Langbein, A. D. Marshall, R. R. Martin. Approximate Congruence Detection of Model Features for Reverse Engineering. In: M.-S. Kim (ed), Proc. International Conference on Shape Modelling and Applications, IEEE Computer Society, Los Alamitos, CA, USA, pp. 69-77. 2003.

[GenComp] Generative Components. Bentley Systems Inc. http://www.generativecomponents.com/. Sept. 28, 2009.

[GML] Generative Modeling Language. http://www.generative-modeling.org/. Sept. 28, 2009.

[Gomes et al. 1998] J. Gomes, L. Darsa, B. Costa, and L. Velho. Warping and morphing of graphical objects. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998.

[Guskov et al. 1999] I. Guskov, W. Sweldens and P. Schröder. Multiresolution signal processing for meshes. In Proceedings of SIGGRAPH 99, 325-334. 1999.

[Guskov et al. 2000] I. Guskov, K. Vidimce, W. Sweldens, and P. Schröder. Normal meshes. In Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques. ACM Press/Addison-Wesley Publishing Co., New York, NY, 95-102. 2000.

[Hable and Rossignac 2007] J. Hable and J. Rossignac. CST: Constructive Solid Trimming for rendering BReps and CSG. IEEE Transactions on Visualization and Computer Graphics, 13(5):1004-1014, 2007.

[Held 2001] M. Held. FIST: Fast Industrial-Strength Triangulation of Polygons. Algorithmica 30(4): 563-596. 2001.

[Hoppe et al. 1992] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In Proceedings of the 19th Annual Conference on Computer Graphics and interactive Techniques J. J. Thomas, Ed. SIGGRAPH '92. ACM, New York, NY, 71-78. 1992.

[Ilic and Fua 2003] S. Ilic and P. Fua. Implicit meshes for modeling and reconstruction. In 2003 Conference on Computer Vision and Pattern Recognition (CVPR 2003), 483-492. 2003.

[Ilic and Fua 2006] S. Ilic and P. Fua. Implicit Meshes for Surface Reconstruction. IEEE Trans. Pattern Anal. Mach. Intell. 28, 2 (Feb. 2006), 328-333. 2006.

[Illustrator] Adobe Illustrator. Adobe Systems Inc. http://www.adobe.com/products/illustrator/. Sept. 28, 2009.

[IronCAD] IronCAD. http://www.ironcad.com/. Sept. 28, 2009.

[Jama] JAMA: A Java Matrix Package. http://math.nist.gov/javanumerics/jama/. Feb. 16, 2010.

[Jang and Rossignac 2007] Justin Jang and Jarek Rossignac. Multiple Object Selection in Pattern Hierarchies. GVU Tech Report GIT-GVU-07-15, 2007.

[Jang and Rossignac 2008] Justin Jang and Jarek Rossignac. OCTOR: OCcurrence selecTOR in pattern hierarchies. In Proceedings of IEEE International Conference on Shape Modeling and Applications (SMI 2008), 205-212, 2008.

[Jang and Rossignac 2009] Justin Jang and Jarek Rossignac. OCTOR: Subset selection in recursive pattern hierarchies. Graphical Models, Volume 71, Issue 2, March 2009, 92-106. 2009.

[Joshi et al. 2007] P. Joshi, M. Meyer, T. DeRose, B. Green, and T. Sanocki. Harmonic coordinates for character articulation. ACM Trans. Graph. 26, 3 (Jul. 2007), 71. 2007.

[Ju et al. 2005] T. Ju, S. Schaefer, and J. Warren. Mean value coordinates for closed triangular meshes. Proceedings of ACM SIGGRAPH 2005, ACM Transactions on Graphics, 24(3):561-566. 2005.

[Kanai et al. 1999] Takashi Kanai, Hiromasa Suzuki, Jun Mitani, Fumihiko Kimura. Interactive Mesh Fusion Based on Local 3D Metamorphosis. Proc. Graphics Interface '99 (Kingston, Canada), pp.148-156, June 1999.

[Kass et al. 1988] KASS, M., WITKIN, A., AND TERZOPOULOS, D. Snakes: Active contour models. International Journal of Computer Vision 1, 4, 321–331. 1988.

[Katz and Tal 2003] Sagi Katz and Ayellet Tal. Hierarchical Mesh Decomposition using Fuzzy Clustering and Cuts. SIGGRAPH 2003, ACM Transactions on Graphics, Volume 22 , Issue 3, July 2003, 954-961. 2003.

[Kim and Rossignac 2003] ByungMoon Kim and Jarek Rossignac. Collision Prediction for Polyhedra under Screw Motions. ACM Symposium in Solid Modeling and Applications, pp. 4-10, June 2003.

[Kim and Rossignac 2005] B. Kim and J. Rossignac. GeoFilter: Geometric Selection of Mesh Filter Parameters. Computer Graphics Forum 24, 3 (2005), 295–302. 2005.

[Kobbelt 2000] L. Kobbelt. Discrete fairing and variational subdivision for freeform surface design . The Visual Computer, 16(3-4):142–158, 2000.

[Kobbelt et al. 1998] L. Kobbelt, S. Campagna, J. Vorsatz, and H. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In Proceedings of the 25th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '98. ACM, New York, NY, 105-114. 1998.

[Kobbelt et al. 1999] Leif Kobbelt, Jens Vorsatz, Hans-Peter Seidel. Multiresolution hierarchies on unstructured triangle meshes. Comput. Geom. 14(1-3): pp. 5-24, 1999.

[Kolomenkin et al. 2009] Michael Kolomenkin, Ilan Shimshoni, and Ayellet Tal. On Edge Detection on Surfaces. In CVPR 2009.

[Krishnamurthy and Levoy 1996] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In Proceedings of the 23rd Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '96. ACM, New York, NY, 313-324. 1996.

[Kuriyama and Kaneko 1999] S. Kuriyama and T. Kaneko. Discrete parameterization for deforming arbitrary meshes. In Proceedings of the 1999 Conference on Graphics interface '99, 132-139. 1999.

[Langbein 2003] F. C. Langbein. Beautification of Reverse Engineered Geometric Models, PhD Thesis. Department of Computer Science, Cardiff University, June 2003.

[Langbein et al. 2001] F. C. Langbein, B. I. Mills, A. D. Marshall, and R. R. Martin. Recognizing Geometric Patterns for Beautification of Reconstructed Solid Models. In Proc. International Conference on Shape Modelling and Applications, Genova, Italy, 7-11 May, 2001, IEEE Computer Society, Los Alamitos, CA, USA, pp. 10-19, 2001.

[Lee and Kim 1998] Jae Yeol Lee and Kwangsoo Kim. A feature-based approach to extracting machining features. Computer-Aided Design 30(13), 1019-1035, 1998.

[Lee et al. 1998] A. W. F. Lee and W. Sweldens and P. Schroeder and L. Cowsar and D. Dobkin. MAPS: Multiresolution Adaptive Parameterization of Surfaces. Computer Graphics Proceedings (SIGGRAPH 98), 95-104. 1998.

[Lee et al. 2000] Lee, A., Moreton, H., and Hoppe, H. Displaced subdivision surfaces. In Proceedings of the 27th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques, 85-94. 2000.

[Lee et al. 2004] Y. Lee, S. Lee, A. Shamir, D. Cohen-Or, and H. Seidel. Intelligent Mesh Scissoring Using 3D Snakes. In Proceedings of the Computer Graphics and Applications, 12th Pacific Conference (October 06 - 08, 2004). PG. IEEE Computer Society, Washington, DC, 279-287. 2004.

[Levy 2001] B. Lévy. Constrained texture mapping for polygonal meshes. In Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '01. ACM, New York, NY, 417-424. 2001.

[Liepa 2003] P. Liepa. Filling holes in meshes. In Symposium on Geometry Processing, 200-205. 2003.

[Lin and Gottschalk 1998] M. C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In In Proceedings of IMA Conference on Mathematics of Surfaces, volume 1, pages 602–608, May 1998.

[Lindenmayer 1968] A. Lindenmayer. Mathematical models for cellular interaction in development, Parts I and II. Journal of Theoretical Biology, 18:280–315, 1968.

[Lipman et al. 2008] Y. Lipman, D. Levin, and D. Cohen-Or. Green Coordinates. ACM Trans. Graph. 27, 3 (Aug. 2008), 1-10. 2008.

[Litke et al. 2001] N. Litke, A. Levin, and P. Schroeder. Fitting subdivision surfaces. In Proceedings of the Conference on Visualization '01 (San Diego, California, October 21 - 26, 2001). VISUALIZATION. IEEE Computer Society, Washington, DC, 319-324. 2001.

[Liu et al. 2006] S. Liu, R. R. Martin, F. C. Langbein, P. L. Rosin. Segmenting Reliefs on Triangle Meshes. In: Proc. ACM Symp. Solid and Physical Modeling, pp. 7-16, 2006.

[Liu et al. 2007a] S. Liu, R. R. Martin, F. C. Langbein, P. L. Rosin. Segmenting Geometric Reliefs from Textured Background Surfaces. Computer-Aided Design and Applications, 4(5):565-583, 2007.

[Liu et al. 2007b] S. Liu, R. R. Martin, F. C. Langbein, P. L. Rosin. Segmenting Periodic Reliefs on Triangle Meshes. In: R. R. Martin, M. A. Sabin, J. J. Winkler (eds), Maths of Surfaces XII, Springer LNCS 4647, 2007.

[Lucas et al. 2005] J.F. Lucas, D.A. Bowman, J. Chen, and C.A. Wingrave. Design and Evaluation of 3D Multiple Object Selection Techniques. Proc. of the ACM I3D, March 2005.

[Ma and Mann 2001] M. Ma and S. Mann. Multiresolution editing of pasted surfaces. In Mathematical Methods for Curves and Surfaces, Oslo 2000, 273-282. 2001.

[Ma et al. 2006] Zhanguo Ma, Bo Liu, and Hongbin Zhang. Cut-and-Paste Editing over 3D Meshes. First International Conference on Innovative Computing, Information and Control, ICICIC '06. Volume 3, 30-01, Aug. 2006, 589-592.

[Maillot et al. 1993] J. Maillot, H. Yahia., and A. Verroust. Interactive texture mapping. In Proceedings of the 20th Annual Conference on Computer Graphics and interactive Techniques (Anaheim, CA, August 02 - 06, 1993). SIGGRAPH '93. ACM, New York, NY, 27-34. 1993.

[Marcheix and Pierra 2002] D. Marcheix and G.A. Pierra. Survey of the persistent naming problem. In Proc. of the Seventh ACM Symposium on Solid Modeling and Applications, Saarbrücken, Germany, June 17-21, 2002.

[Masuda et al. 2004] Hiroshi Masuda, Yoshiyuki Furukawa, Yasuhiro Yoshioka. Volume-Based Cut-and-Paste Editing For Early Design Phases. ASME Design

Engineering Technical Conferences and Computers and Information in Engineering Conference, September 28-October 2, 2004, Salt Lake City, Utah USA.

[MathForum] Orthogonal Distance Regression Planes. http://mathforum.org/library/drmath/view/63765.html. April 3, 2010.

[Maya] Autodesk Maya. http://www.autodesk.com/maya. Sept. 28, 2009.

[MicroStation] MicroStation. Bentley Systems Inc. http://www.bentley.com/microstation. Sept. 28, 2009.

[Miller and Myers 2002] Robert C. Miller and Brad A. Myers. Multiple Selections in Smart Text Editing. Proc. of the 6th International Conference on Intelligent User Interfaces (IUI 2002), San Francisco, CA, Jan. 2002, pp 103-110.

[Mills et al. 2001] B. I. Mills, F. C. Langbein, A. D. Marshall, and R. R. Martin. Approximate Symmetry Detection for Reverse Engineering. In: D. C. Anderson, K. Lee (eds), Proc. 6th ACM Symp. Solid Modelling and Applications, pp. 241-248. 2001.

[Mokhtarian et al. 1998] F. Mokhtarian, N. Khalili, and P. Yuen. Multi-Scale 3-D Free-Form Surface Smoothing. In Proc. British Machine Vision Conference, 730-739. 1998.

[Mortenson 2007] M. Mortenson. Geometric Transformations for 3D Modeling. Industrial Press Inc., New York, 2007.

[Mueller et al. 2006] Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, Luc Van Gool. Procedural Modeling of Buildings. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006), volume 25, number 3, 614-623. 2006.

[Nackman et al. 1986] L. R. Nackman, M. A. Lavin, R. H. Taylor, W. C. Dietrich, and D. D. Grossman. AML/X: a programming language for design and manufacturing. In Proceedings of 1986 ACM Fall Joint Computer Conference (Dallas, Texas, United States). IEEE Computer Society Press, Los Alamitos, CA, 145-159. 1986.

[NX] NX. Siemens Product Lifecycle Management Software Inc. http://www.plm.automation.siemens.com/en_us/products/nx/. Sept. 28, 2009.

[O'Mara 2002] O'Mara, David Thomas John. Automated Facial Metrology. Ph.D. Thesis, University of Western Australia, 2002.

[Oh et al. 2006] Ji-Young Oh, W. Stuerzlinger, and D. Dadgari. Group Selection Techniques for Efficient 3D Modeling. IEEE Symposium on 3D User Interfaces (3DUI 2006), University of Arizona, Tucson, March 25-29, pp 95-102, 2006.

[Onishi et al. 2003] K. Onishi, S. Hasuike, Y. Kitamura, and F. Kishino. Interactive modeling of trees by using growth simulation. In Proceedings of the ACM

Symposium on Virtual Reality Software and Technology (Osaka, Japan, October 01-03, 2003). VRST '03. ACM, New York, NY, 66-72. 2003.

[OSG] Open Scene Graph. http://www.openscenegraph.org/. Sept. 28, 2009.

[Parish and Mueller 2001] Y. I. Parish and P. Mueller. Procedural modeling of cities. In Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '01. ACM, New York, NY, 301-308. 2001.

[Pauly and Gross 2001] M. Pauly and M. Gross. Spectral processing of point-sampled geometry. In Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques SIGGRAPH '01. ACM, New York, NY, 379-386. 2001.

[Pauly et al. 2008] Pauly, M., Mitra, N. J., Wallner, J., Pottmann, H., and Guibas, L. J. 2008. Discovering structural regularity in 3D geometry. In ACM SIGGRAPH 2008 Papers (Los Angeles, California, August 11 - 15, 2008). SIGGRAPH '08. ACM, New York, NY, 1-11.

[Perez et al. 2003] P. Perez, M. Gangnet, and A. Blake. Poisson image editing. ACM Trans. Graph. 22, 3, 313-318. 2003.

[Pernot et al. 2003b] J. Pernot, G. Moraru, and P. Véron. Filling holes in meshes using a mechanical model to simulate the curvature variation minimization. Computers and Graphics, 30, 6 (Dec. 2006), 892-902. 2006.

[POV-Ray] POV-Ray Documentation. Persistence of Vision Raytracer, Pty. Ltd. http://www.povray.org/documentation/. Sept. 28, 2009.

[Power et al. 1999] Joanna L. Power, A. J. Bernheim Brush, Przemyslaw Prusinkiewicz, and David H. Salesin: Interactive arrangement of botanical L-system models. In Proceedings of the 1999 Symposium on Interactive 3D Graphics, pp. 175-182 and 234. 1999.

[PowerPoint] Microsoft Office PowerPoint. Microsoft Corporation. http://office.microsoft.com/powerpoint. Sept. 28, 2009.

[Praun et al. 2000] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. ACM SIGGRAPH 2000 Conference Proceedings, 465-470. 2000.

[ProE] Pro/ENGINEER. Parametric Technology Corporation. http://www.ptc.com/products/proengineer/. Sept. 28, 2009.

[Prusinkiewicz and Lindenmayer 1990] P. Prusinkiewicz and A. Lindenmayer. The Algorithmic Beauty of Plants. Springer-Verlag, New York, 1990.

[Prusinkiewicz et al. 1994] P. Prusinkiewicz, M. James and R. Mech. Synthetic Topiary. In Proc. ACM SIGGRAPH '94, 351-358. 1994.

[Prusinkiewicz et al. 2001] Przemyslaw Prusinkiewicz, Lars Muendermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. Proceedings of SIGGRAPH 2001 (Los Angeles, California, August 12-17, 2001), 289-300. 2001.

[Rappoport 1993] Ari Rappoport. A scheme for single instance representation in hierarchical assembly graphs. B. Falcidieno, T.L. Kunii (eds), Geometric Modeling in Computer Graphics, pp. 213-224, 1993.

[Reed et al. 1995] M. Reed, P. K. Allen, and S. Abrams. CAD model acquisition using BSP trees. In Proc. IROS International Conference on Intelligent Robots and Systems, pp. 335-339, August 1995.

[Regli et al. 1994] W. Regli, S. Gupta, and D. Nau. Feature recognition for manufacturability analysis. Tech. Rep. ISR TR94-10, University of Maryland, February 1994.

[Rhino3D] Rhinoceros 3D. Robert McNeel & Associates. http://www.rhino3d.com/. Sept. 28, 2009.

[Rogers 1985] David Rogers. Procedural Elements for Computer Graphics. McGraw-Hill, New York, New York. 1985.

[Rossignac 1986] Jarek Rossignac. Constraints in Constructive Solid Geometry. Proc. ACM Workshop on Interactive 3D Graphics, ACM Press, pp. 93-110, Chapel Hill, 1986.

[Rossignac and Kim 2001] J. Rossignac and J. Kim. Computing and visualizing pose-interpolating 3-D motions. Journal of Computer-Aided Design, vol. 33, no. 4, pp. 279-291, April 2001.

[Rossignac and Vinacua 2009] J. Rossignac and A. Vinacua. Extraction of Affinity Roots for Computing Steady Affine Morphs in 3D. Work in progress. 2009.

[Rossignac et al. 1988] Jarek Rossignac, Paul Borrel, and Lee Nackman. Interactive Design with Sequences of Parameterized Transformations. Proc. 2nd Eurographics Workshop on Intelligent CAD Systems: Implementation Issues, April 11-15, Veldhoven, The Netherlands, pp. 95-127, 1988.

[Savchenko and Kojekine 2002] V. Savchenko and N. Kojekine. An approach to blend surfaces. In CGI, 139-150. 2002.

[Schneider and Kobbelt 2001] Robert Schneider and Leif Kobbelt. Geometric Fairing of Irregular Meshes for Free-Form Surface Design. Computer Aided Geometric Design (CAGD), Volume 18, Issue 4, 359-379. 2001.

[Schneider et al. 2001] R. Schneider, L. Kobbelt, and H. Seidel. Improved bi-Laplacian mesh fairing. In Mathematical Methods For Curves and Surfaces: Oslo 2000, T.

Lyche and L. L. Schumaker, Eds. 1 ed. Vanderbilt Univ. Press Innovations In Applied Mathematics Series. Vanderbilt University, Nashville, TN, 445-454. 2001.

[Shah and Mantyla 1995] J. Shah and M. Mantyla. Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications. John Wiley & Sons, Inc.: New York, NY. 1995.

[Shamir 2008] A. Shamir. A survey on mesh segmentation techniques. Computer Graphics Forum, 27, 6, 1539-1556. 2008.

[Sharf et al. 2004] Andrei Sharf, Marc Alexa and Daniel Cohen-Or. Context-based Surface Completion. SIGGRAPH 2004.

[Sharf et al. 2006] Andrei Sharf, Marina Blumenkrants, Ariel Shamir, and Daniel Cohen-Or. SnapPaste: An Interactive Technique for Easy Mesh Composition. The Visual Computer: International Journal of Computer Graphics, Vol. 22, Issue 9 (September 2006), 835-844, 2006.

[Sheffer and deSturler 2000] Alla Sheffer and Eric de Sturler. Surface parameterization for meshing by triangulation flattening. Proc. 9th Int. Meshing Roundtable, Sandia Nat. Lab., 161–172. 2000.

[Sheffer et al. 2006] Sheffer, A., Praun, E., and Rose, K. Mesh parameterization methods and their applications. Found. Trends. Comput. Graph. Vis. 2, 2 (Jan. 2006), 105-171. 2006.

[Smith 1984] A. R. Smith. Plants, fractals, and formal languages. SIGGRAPH Comput. Graph. 18, 3 (Jul. 1984), pp. 1-10. 1984.

[Snyder and Kajiya 1992] John M. Snyder and James T. Kajiya. Generative Modeling: A Symbolic System for Geometric Modeling. In Proceedings of SIGGRAPH 1992, pp. 369-378. 1992.

[SolidWorks] SolidWorks. Dassault Systems SolidWorks Corp. http://www.solidworks.com/. Sept. 28, 2009.

[Song et al. 2004] Y. Song, J. S. M. Vergeest, and S. Spanjaard. Fitting and Manipulating Freeform Shapes by Extendable Freeform Templates. In Proceedings of the Shape Modeling international 2004 (June 07 - 09, 2004). Shape Modeling International. IEEE Computer Society, Washington, DC, 43-52. 2004.

[Sorkine et al. 2004] Olga Sorkine, Daniel Cohen-Or, Yaron Lipman, Marc Alexa, Christian Rössl and Hans-Peter Seidel. Laplacian Surface Editing. Eurographics/ACM SIGGRAPH Symposium on Geometry Processing 2004.

[Stiny and Gips 1971] G. Stiny and J. Gips. Shape grammars and the generative specification of painting and sculpture. IFIP Congress 1971. North Holland Publishing Co. 1971.

[Surazhsky and Gotsman 2003] V. Surazhsky and C. Gotsman. Explicit surface remeshing. In Proceedings of the 2003 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (Aachen, Germany, June 23 - 25, 2003). ACM International Conference Proceeding Series, vol. 43, 20-30. 2003.

[Suzuki et al. 2000] H. Suzuki, Y. Sakurai, T. Kanai, and F. Kimura. Interactive mesh dragging with an adaptive remeshing technique. The Visual Computer, 16(3-4):159--176, 2000.

[Szymczak et al. 2002] Andrzej Szymczak, Davis King and Jarek Rossignac. Piecewise Regular Meshes: Construction and Compression. Graphical Models 64(3-4), 183-198, 2002.

[Tasdizen et al. 2002] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher. Geometric Surface Smoothing via Anisotropic Diffusion of Normals. In Proceedings, IEEE Visualization 2002, 125-132. 2002.

[Taubin 1995] G. Taubin. A signal processing approach to fair surface design. In Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques S. G. Mair and R. Cook, Eds. SIGGRAPH '95. ACM, New York, NY, 351-358. 1995.

[Thompson et al. 1999] W.B. Thompson, J.C. Owen, H.J. de St. Germain, S.R. Stark Jr., and T.C. Henderson. Feature-based reverse engineering of mechanical parts. IEEE Transactions on Robotics and Automation, 15(1), pp. 57-66. 1999.

[Traband et al. 1996] M. T. Traband, F. W. Tillotson, and J. D. Martin. Reverse and re-engineering in the DOD organic maintenance community: Current status and future direction. Tech. Rep. 96-060, Applied Research Laboratory, The Pennsylvania State University, February 1996.

[Tsang 1998] Clara Tsang. Animated Surface Pasting. Research Report CS-98-19, Masters Thesis, University of Waterloo, Waterloo, Ontario, August 1998.

[Turk 1992] G. Turk. Re-tiling polygonal surfaces. SIGGRAPH Comput. Graph. 26, 2 (Jul. 1992), 55-64. 1992.

[Vallet and Levy 2008] Bruno Vallet and Bruno Levy. Spectral Geometry Processing with Manifold Harmonics. Computer Graphics Forum (Proceedings Eurographics), 2008.

[van Emmerik et al. 1993] Martin van Emmerik, Ari Rappoport, and Jarek Rossignac. Simplifying interactive design of solid models: A hypertext approach. The Visual Computer, vol. 9, No. 5, pp. 239-254, March 1993.

[van Wijk 1986] J. J. van Wijk. SML: a solid modelling language. Computer Aided Design, 18, 10 (Oct. 1986), 443-449. 1986.

[Vandenbrande and Requicha 1993] J. Vandenbrande and A. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. IEEE Trans. on Pattern Analysis and Machine Intelligence, pp. 1269-1285, December 1993.

[vanderMeiden 2008] Hilderick A. van der Meiden. Semantics of Families of Objects. PhD Thesis, Delft University of Technology, The Netherlands. 2008.

[Várady et al. 1990] T. Várady, B. Gaál, and G. E. M. Jared. Identifying features in solid modelling. Computers in Industry, (Joe Hatvany Memorial Issue), Vol 14, No 1-3, pp 43-51. 1990.

[Várady et al. 1997] T. Várady, R. R. Martin, and J. Cox. Reverse Engineering of Geometric Models - An Introduction. Computer-Aided Design, 29 (4), pp 255-269. 1997.

[Verdera et al. 2003] J. Verdera, V. Caselles, M. Bertalmio, and G. Sapiro. Inpainting surface holes. In 2003 International Conference on Image Processing (ICIP), 903-906. 2003.

[Vorsatz et al. 2003] J. Vorsatz, C. Rössl, and H. Seidel. Dynamic remeshing and applications. In Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications (Seattle, Washington, USA, June 16 - 20, 2003). SM '03. ACM, New York, NY, 167-175. 2003.

[Weber et al. 2009] O. Weber, M. Ben-Chen, and C. Gotsman. Complex Barycentric Coordinates with Applications to Image Deformation. Computer Graphics Forum, 28(2), 2009.

[Weyl 1982] H. Weyl. Symmetry. Princeton University Press, Princeton, New Jersey, 1982.

[Wonka et al. 2003] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky. Instant architecture. ACM Trans. Graph. 22, 3 (Jul. 2003), 669-677. 2003.

[Yu et al. 2004] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, Heung-Yeung Shum. Mesh Editing with Poisson-Based Gradient Field Manipulation. ACM Trans. on Graphics (In Proceedings of ACM SIGGRAPH 2004), 2004.

[Zatzarinni et al. 2009] Rony Zatzarinni, Ayellet Tal, and Ariel Shamir. Relief Analysis and Extraction. SIGGRAPH Asia 2009, ACM Transactions on Graphics, Volume 28, Issue 5, December 2009.

[Zelinka and Garland 2006] S. Zelinka and M. Garland. Surfacing by numbers. In Proceedings of Graphics interface 2006. ACM International Conference Proceeding Series, vol. 137, 107-113. 2006.

[Zhang et al. 2005] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Feature-Based Surface Parameterization and Texture Mapping. ACM Transaction on Graphics, Vol 24(1), 1-27. 2005.

[Zitová and Flusser 2003] Barbara Zitová and Jan Flusser. Image registration methods: a survey. Image Vision Comput. 21(11): 977-1000. 2003.

[Zwicker et al. 2002] M. Zwicker, M. Pauly, O. Knoll, and M. Gross. Pointshop 3D: an interactive system for point-based surface editing. In Proceedings of the 29th Annual Conference on Computer Graphics and interactive Techniques (San Antonio, Texas, July 23 - 26, 2002). SIGGRAPH '02. ACM, New York, NY, 322-329. 2002.

VITA

Justin Jang was born in Boston, Massachusetts, U.S.A. After completing a bachelors in Computer Engineering at Auburn University, he entered the Ph.D program at the Georgia Institute of Technology. He has explored a wide variety of topics in computer graphics including volume visualization and mesh simplification and has authored papers on these topics. Under the guidance of Dr. Jarek Rossignac, he is exploring the design and editing of geometric patterns and geometric relief feature transfer.